Source Code For Angio-Based Diameter Measurement System

CANVAS Group
Authors: Meide Zhao, PhD, mzhao@uic.edu
September, 1999

```
//
//  CANVAS COPYRIGHT, ALL RIGHTS RESERVED, September, 1999.
//
//  ADMS  (Angio-based Diameter Measurement System)
//
//  Authors: Meide Zhao, PhD, mzhao@uic.edu
//

////////////////////////////////////////////////////////////////////
// This is a driver ViewKit program generated by RapidApp 1.2
//
// This program instantiates a ViewKit VkApp object and creates
// any main window objects that are meant to be shown at startup.
// Although editable code blocks are provided, there should rarely.
// be any reason to modify this file. Make application-specific
// changes in the classes created by the main window classes
// You can add also additional initialization in subclasses of VkApp
////////////////////////////////////////////////////////////////////
#include <Vk/VkApp.h>

// Headers for window classes used in this program

#include "MagicWinMainWindow.h"

//---- Start editable code block: headers and declarations


//---- End editable code block: headers and declarations


// Fallback resources

static char *fallbackResources[] = {
    "*useSchemes:     all",
    "*sgiMode:        true",
    "*useEnhancedFSB: true",
    "*keyboardFocusPolicy: explicit",
    "*magicWin*title:  CANVAS",
    //---- Start editable code block: fallbacks

    //---- End editable code block: fallbacks
    NULL
};


void main ( int argc, char **argv )
{
    extern void InitEZ(void);

    InitEZ(); // Only need to force bind EZ library              // for
Fix+Continue

    //---- Start editable code block: main initialization

    //---- End editable code block: main initialization

    VkApp::setFallbacks(fallbackResources);

    VkApp       *app;
```

```
    // Create an application object

    app = new VkApp("Magic", &argc, argv);

    //---- Start editable code block:  post init

    //---- End editable code block:  post init

    // Create the top level windows

    VkSimpleWindow *magicWin  = new MagicWinMainWindow("magicWin");
    magicWin->show();

    //---- Start editable code block:  event loop

    //---- End editable code block:  event loop

    app->run ();

}

//---- Start editable code block: End of generated code


//---- End editable code block: End of generated code
```

```
    //---- End editable code block: CalibBB protected

  private:
    static void* RegisterCalibBBInterface();
    //---- Start editable code block: CalibBB private

    //---- End editable code block: CalibBB private

};
//---- Start editable code block: End of generated code

//---- End editable code block: End of generated code
#endif
```

```
////////////////////////////////////////////////////////////
//
// Header file for CalibBB
//
//    This file is generated by RapidApp 1.2
//
//    This class is derived from CalibBBUI which
//    implements the user interface created in
//    RapidApp. This class contains virtual
//    functions that are called from the user interface.
//
//    When you modify this header file, limit your changes to those
//    areas between the "//---- Start/End editable code block" markers
//
//    This will allow RapidApp to integrate changes more easily
//
//    This class is a ViewKit user interface "component".
//    For more information on how components are used, see the
//    "ViewKit Programmers' Manual", and the RapidApp
//    User's Guide.
////////////////////////////////////////////////////////////
#ifndef CALIBBB_H
#define CALIBBB_H
#include "CalibBBUI.h"
//---- Start editable code block: headers and declarations


//---- End editable code block: headers and declarations


//---- CalibBB class declaration

class CalibBB : public CalibBBUI
{

  public:

    CalibBB ( const char *, Widget );
    CalibBB ( const char * );
    ~CalibBB();
    const char *  className();

    static VkComponent *CreateCalibBB( const char *name, Widget parent );

    //---- Start editable code block: CalibBB public

    class VkComponent *_parent;
    void set(class VkComponent *v) {_parent = v;}


    //---- End editable code block: CalibBB public



  protected:


    // These functions will be called as a result of callbacks
    // registered in CalibBBUI

    virtual void calibAccept1 ( Widget, XtPointer );

    //---- Start editable code block: CalibBB protected
```

```
    //      void memberFunction ( Type );
    //
    // where "Type" is one of:
    //     const char *     (Use XmRString)
    //     Boolean          (Use XmRBoolean)
    //     int              (Use XmRInt)
    //     float            (Use XmRFloat)
    //     No argument      (Use VkRNoArg or "NoArg"
    //     A filename       (Use VkRFilename or "Filename")
    //     An enumeration   (Use "Enumeration:ClassName:Type: VALUE1, VALUE2, VALUE3")
    //     A callback       (Use XmRCallback)


    static InterfaceMap map[] = {
    //---- Start editable code block: CalibBBUI resource table

      // { "resourceName", "setAttribute", XmRString},
    //---- End editable code block: CalibBBUI resource table
      { NULL }, // MUST be NULL terminated
    };

    return map;
} // End RegisterCalibBBInterface()



//---- End of generated code

//---- Start editable code block: End of generated code


//---- End editable code block: End of generated code
```

```
    printf(" Pixsize:\n");

    float d = atof(XmTextFieldGetString(_textfieldCalibDistance));
    printf(" Pixsize: %f \n", d);

    float p = atof(XmTextFieldGetString(((MagicDeckTabbedDeck *)_parent) -> get_textfie

    char str[50];
    if(p != 0.0)
    {
      sprintf(str, "%8.3f", d/p);

      printf(" Pixsize:    %f %f %f\n", p, d, d/p);
      XmTextFieldSetString(_textfieldPixsize, str);
    }

    //---- End editable code block: CalibBB calibAccept1

}    // End CalibBB::calibAccept1()




//////////////////////////////////////////////////////////////////////
// static creation function, for importing class into rapidapp
// or dynamically loading, using VkComponent::loadComponent
//////////////////////////////////////////////////////////////////////


VkComponent *CalibBB::CreateCalibBB( const char *name, Widget parent )
{
    VkComponent *obj =  new CalibBB ( name, parent );
    return ( obj );
} // End CreateCalibBB


//////////////////////////////////////////////////////////////////////
// Function for accessing a description of the dynamic interface
// to this class.
//////////////////////////////////////////////////////////////////////


// WARNING: This structure is different than that used with 1.1 RapidApp.
// See the RapidApp release notes for details

struct InterfaceMap {
  char  *resourceName;
  char  *methodName;
  char  *argType;
  char  *definingClass; // Optional, if not this class
  void (VkCallbackObject::*method)(...); // Reserved, do not set
};


void *CalibBB::RegisterCalibBBInterface()
{
    // This structure registers information about this class
    // that allows RapidApp to create and manipulate an instance.
    // Each entry provides a resource name that will appear in the
    // resource manager palette when an instance of this class is
    // selected, the name of the member function as a string,
    // the type of the single argument to this function, and an.
    // optional argument indicating the class that defines this function.
    // All member functions must have the form
    //
```

```
    // which calls CalibBBUI::create() to create
    // the widgets for this component. Any code added here
    // is called after the component's interface has been built

    //---- Start editable code block: CalibBB constructor


    //---- End editable code block: CalibBB constructor



}    // End Constructor



CalibBB::CalibBB(const char *name) :
                  CalibBBUI(name)
 {
    // This constructor calls CalibBBUI(name)
    // which does not create any widgets. Usually, this
    // constructor is not used

    //---- Start editable code block: CalibBB constructor 2


    //---- End editable code block: CalibBB constructor 2



}    // End Constructor



CalibBB::~CalibBB()
{
    // The base class destructors are responsible for
    // destroying all widgets and objects used in this component.
    // Only additional items created directly in this class
    // need to be freed here.

    //---- Start editable code block: CalibBB destructor


    //---- End editable code block: CalibBB destructor



}    // End Destructor


const char * CalibBB::className() // classname
{
    return ("CalibBB");
} // End className()


void CalibBB::calibAccept1 ( Widget w, XtPointer callData )
{
    //---- Start editable code block: CalibBB calibAccept1

    XmPushButtonCallbackStruct *cbs = (XmPushButtonCallbackStruct*) callData;

    //--- Comment out the following line when CalibBB::calibAccept1 is implemented:

    //::VkUnimplemented ( w, "CalibBB::calibAccept1" );
```

```
////////////////////////////////////////////////////////////////
//
// Source file for CalibBB
//
//    This file is generated by RapidApp 1.2
//
//    This class is derived from CalibBBUI which
//    implements the user interface created in
//    RapidApp. This class contains virtual
//    functions that are called from the user interface.
//
//    When you modify this source, limit your changes to
//    modifying the sections between the
//    "//---- Start/End editable code block" markers
//
//    This will allow RapidApp to integrate changes more easily
//
//    This class is a ViewKit user interface "component".
//    For more information on how components are used, see the
//    "ViewKit Programmers' Manual", and the RapidApp
//    User's Guide.
////////////////////////////////////////////////////////////////

#include "CalibBB.h"
#include <Vk/VkEZ.h>
#include <Xm/BulletinB.h>
#include <Xm/Label.h>
#include <Xm/PushB.h>
#include <Xm/TextF.h>
#include <Vk/VkResource.h>


extern void VkUnimplemented ( Widget, const char * );

////////////////////////////////////////////////////////////////////////////
// The following non-container elements are created by CalibBBUI and are
// available as protected data members inherited by this class
//
//  XmPushButton                    _calibAccept
//  XmLabel                 _labelCalib2
//  XmLabel                 _labelCalib1
//  XmLabel                 _labelCalibPixsize
//  XmLabel                 _labelCalibDistance
//  XmTextField             _textfieldCalibDistance
//  XmTextField             _textfieldPixsize
//
////////////////////////////////////////////////////////////////////////////


//---- Start editable code block: headers and declarations

#include "MagicDeckTabbedDeck.h"
#include <stdio.h>
#include <math.h>

//---- End editable code block: headers and declarations



//---- CalibBB Constructor

CalibBB::CalibBB(const char *name, Widget parent) :
                    CalibBBUI(name, parent)
{
    // This constructor calls CalibBBUI(parent, name)
```

```
    // These virtual functions are called from the private callbacks (below)
    // Intended to be overridden in derived classes to define actions

    virtual void calibAccept1 ( Widget, XtPointer );

    //---- Start editable code block: CalibBB protected


    //---- End editable code block: CalibBB protected


  private:

    // Array of default resources

    static String      _defaultCalibBBUIResources[];


    // Callbacks to interface with Motif

    static void calibAccept1Callback ( Widget, XtPointer, XtPointer );

    //---- Start editable code block: CalibBB private

    friend class MagicDeckTabbedDeck;

    //---- End editable code block: CalibBB private
};
//---- Start editable code block: End of generated code


//---- End editable code block: End of generated code

#endif
```

```
////////////////////////////////////////////////////////////
//
// Header file for CalibBBUI
//
//    This file is generated by RapidApp 1.2
//
//    This class implements the user interface portion of a class
//    Normally it is not used directly.
//    Instead the subclass, CalibBB is instantiated
//
//    To extend or alter the behavior of this class, you should
//    modify the CalibBB files
//
//    Restrict changes to those sections between
//    the "//--- Start/End editable code block" markers
//
//    This will allow RapidApp to integrate changes more easily
//
//    This class is a ViewKit user interface "component".
//    For more information on how components are used, see the
//    "ViewKit Programmers' Manual", and the RapidApp
//    User's Guide.
//
////////////////////////////////////////////////////////////
#ifndef CALIBBBUI_H
#define CALIBBBUI_H
#include <Vk/VkComponent.h>


//---- Start editable code block: headers and declarations


//---- End editable code block: headers and declarations


class CalibBBUI : public VkComponent
{

  public:

    CalibBBUI ( const char *, Widget );
    CalibBBUI ( const char * );
    ~CalibBBUI();
    void create ( Widget );
    const char *  className();

    //---- Start editable code block: CalibBB public

    //---- End editable code block: CalibBB public


  protected:


    // Widgets created by this class

    Widget  _calibAccept;
    Widget  _calibBB;
    Widget  _labelCalib1;
    Widget  _labelCalib2;
    Widget  _labelCalibDistance;
    Widget  _labelCalibPixsize;
    Widget  _textfieldCalibDistance;
    Widget  _textfieldPixsize;
```

```
//---- End editable code block: End of generated code
```

```
    _textfieldCalibDistan[redacted] XtVaCreateManagedWidget  ( [redacted]tfieldCalibDistance",
                                                              xmTextFieldWidgetClass,
                                                              _baseWidget,
                                                              XmNx, 140,
                                                              XmNy, 120,
                                                              XmNwidth, 120,
                                                              XmNheight, 40,
                                                              (XtPointer) NULL );


    _textfieldPixsize = XtVaCreateManagedWidget  ( "textfieldPixsize",
                                                   xmTextFieldWidgetClass,
                                                   _baseWidget,
                                                   XmNx, 140,
                                                   XmNy, 190,
                                                   XmNwidth, 120,
                                                   XmNheight, 40,
                                                   (XtPointer) NULL );


    //---- Start editable code block: CalibBBUI create


    //---- End editable code block: CalibBBUI create
}

const char * CalibBBUI::className()
{
    return ("CalibBBUI");
}    // End className()



////////////////////////////////////////////////////////////////////
// The following functions are static member functions used to
// interface with Motif.
////////////////////////////////////

void CalibBBUI::calibAccept1Callback ( Widget    w,
                                       XtPointer clientData,
                                       XtPointer callData )
{
    CalibBBUI* obj = ( CalibBBUI * ) clientData;
    obj->calibAccept1 ( w, callData );
}



////////////////////////////////////////////////////////////////////
// The following functions are called from the menu items
// in this window.
////////////////////////////////////

void CalibBBUI::calibAccept1 ( Widget, XtPointer )
{
    // This virtual function is called from calibAccept1Callback.
    // This function is normally overriden by a derived class.

}



//---- Start editable code block: End of generated code
```

```
// Create widgets used in this component
// All variables are data members of this class

_calibAccept = XtVaCreateManagedWidget  ( "calibAccept",
                                          xmPushButtonWidgetClass,
                                          _baseWidget,
                                          XmNlabelType, XmSTRING,
                                          XmNx, 140,
                                          XmNy, 270,
                                          XmNwidth, 120,
                                          XmNheight, 40,
                                          (XtPointer) NULL );

XtAddCallback ( _calibAccept,
                XmNactivateCallback,
                &CalibBBUI::calibAccept1Callback,
                (XtPointer) this );


_labelCalib2 = XtVaCreateManagedWidget  ( "labelCalib2",
                                          xmLabelWidgetClass,
                                          _baseWidget,
                                          XmNlabelType, XmSTRING,
                                          XmNx, 280,
                                          XmNy, 200,
                                          XmNwidth, 64,
                                          XmNheight, 20,
                                          (XtPointer) NULL );


_labelCalib1 = XtVaCreateManagedWidget  ( "labelCalib1",
                                          xmLabelWidgetClass,
                                          _baseWidget,
                                          XmNlabelType, XmSTRING,
                                          XmNx, 281,
                                          XmNy, 130,
                                          XmNwidth, 28,
                                          XmNheight, 20,
                                          (XtPointer) NULL );


_labelCalibPixsize = XtVaCreateManagedWidget  ( "labelCalibPixsize",
                                                xmLabelWidgetClass,
                                                _baseWidget,
                                                XmNlabelType, XmSTRING,
                                                XmNx, 11,
                                                XmNy, 200,
                                                XmNwidth, 73,
                                                XmNheight, 20,
                                                (XtPointer) NULL );


_labelCalibDistance = XtVaCreateManagedWidget  ( "labelCalibDistance",
                                                 xmLabelWidgetClass,
                                                 _baseWidget,
                                                 XmNlabelType, XmSTRING,
                                                 XmNx, 11,
                                                 XmNy, 129,
                                                 XmNwidth, 69,
                                                 XmNheight, 20,
                                                 (XtPointer) NULL );
```

```
    // code to a derived class constructor.

    //---- Start editable code block: CalibBB constructor 2


    //---- End editable code block: CalibBB constructor 2

}    // End Constructor


CalibBBUI::CalibBBUI ( const char *name, Widget parent ) : VkComponent ( name )
{
    //---- Start editable code block: CalibBB pre-create


    //---- End editable code block: CalibBB pre-create


    // Call creation function to build the widget tree.
     create ( parent );
    //---- Start editable code block: CalibBB constructor

    //---- End editable code block: CalibBB constructor

}    // End Constructor

CalibBBUI::~CalibBBUI()
{
    // Base class destroys widgets
    //---- Start editable code block: CalibBBUI destructor


    //---- End editable code block: CalibBBUI destructor
}    // End destructor


void CalibBBUI::create ( Widget parent )
{
    Arg      args[6];
    Cardinal count;
    count = 0;

    // Load any class-defaulted resources for this object

    setDefaultResources ( parent, _defaultCalibBBUIResources  );


    // Create an unmanaged widget as the top of the widget hierarchy

    _baseWidget = _calibBB = XtVaCreateWidget ( _name,
                                                xmBulletinBoardWidgetClass,
                                                parent,
                                                XmNresizePolicy, XmRESIZE_GROW,
                                                (XtPointer) NULL );

    // install a callback to guard against unexpected widget destruction
```

```
////////////////////////////////////////////////////////////
//
// Source file for CalibBBUI
//
//    This class implements the user interface created in
//    RapidApp.
//
//    Restrict changes to those sections between
//    the "//--- Start/End editable code block" markers
//
//    This will allow RapidApp to integrate changes more easily
//
//    This class is a ViewKit user interface "component".
//    For more information on how components are used, see the
//    "ViewKit Programmers' Manual", and the RapidApp
//    User's Guide.
//
//
////////////////////////////////////////////////////////////


#include "CalibBBUI.h" // Generated header file for this class

#include <Xm/BulletinB.h>
#include <Xm/Label.h>
#include <Xm/PushB.h>
#include <Xm/TextF.h>
#include <Vk/VkResource.h>
//---- Start editable code block: headers and declarations


//---- End editable code block: headers and declarations



// These are default resources for widgets in objects of this class
// All resources will be prepended by *<name> at instantiation,
// where <name> is the name of the specific instance, as well as the
// name of the baseWidget. These are only defaults, and may be overriden
// in a resource file by providing a more specific resource name

String  CalibBBUI::_defaultCalibBBUIResources[] = {
        "*calibAccept.labelString:  Accept",
        "*labelCalib1.labelString:  mm",
        "*labelCalib2.labelString:  mm/pixel",
        "*labelCalibDistance.labelString:  Distance ",
        "*labelCalibPixsize.labelString:  Pixel Size ",
        "*tabLabel:  Calib",
        "*textfieldCalibDistance.value:  19.0",
        "*textfieldPixsize.value:  0.215",

        //---- Start editable code block: CalibBBUI Default Resources


        //---- End editable code block: CalibBBUI Default Resources

        (char*)NULL
};

CalibBBUI::CalibBBUI ( const char *name ) : VkComponent ( name )
{
    // No widgets are created by this constructor.
    // If an application creates a component using this constructor,
    // It must explictly call create at a later time.
    // This is mostly useful when adding pre-widget creation
```

```
    int  _numH, _numV;
    int  _mode;
    void set_mode(int);

    int _blow;
    void set_blow(int);

    class VkComponent *_draw;
    void set(class VkComponent *v) {_draw = v;}

    //---- End editable code block: DispBB public


  protected:

    // These functions will be called as a result of callbacks
    // registered in DispBBUI

    virtual void blowUpButton ( Widget, XtPointer );
    virtual void blowUpDistance ( Widget, XtPointer );
    virtual void blowUpPan ( Widget, XtPointer );
    virtual void blowUpZoom ( Widget, XtPointer );
    virtual void magicDown ( Widget, XtPointer );
    virtual void magicUp ( Widget, XtPointer );
    virtual void setMode10x10 ( Widget, XtPointer );
    virtual void setMode1x1 ( Widget, XtPointer );
    virtual void setMode1x2 ( Widget, XtPointer );
    virtual void setMode2x1 ( Widget, XtPointer );
    virtual void setMode2x2 ( Widget, XtPointer );
    virtual void setMode3x3 ( Widget, XtPointer );
    virtual void setMode4x4 ( Widget, XtPointer );
    virtual void setMode5x5 ( Widget, XtPointer );
    virtual void setMode6x6 ( Widget, XtPointer );
    virtual void setMode7x7 ( Widget, XtPointer );
    virtual void setMode8x8 ( Widget, XtPointer );
    virtual void setMode9x9 ( Widget, XtPointer );

    //---- Start editable code block: DispBB protected


    //---- End editable code block: DispBB protected


  private:

    static void* RegisterDispBBInterface();

    //---- Start editable code block: DispBB private


    //---- End editable code block: DispBB private


};
//---- Start editable code block: End of generated code


//---- End editable code block: End of generated code

#endif
```

```
//////////////////////////////////////////////////////////////////
//
// Header file for DispBB
//
//    This file is generated by RapidApp 1.2
//
//    This class is derived from DispBBUI which
//    implements the user interface created in
//    RapidApp. This class contains virtual
//    functions that are called from the user interface.
//
//    When you modify this header file, limit your changes to those
//    areas between the "//---- Start/End editable code block" markers
//
//    This will allow RapidApp to integrate changes more easily
//
//    This class is a ViewKit user interface "component".
//    For more information on how components are used, see the
//    "ViewKit Programmers' Manual", and the RapidApp
//    User's Guide.
//////////////////////////////////////////////////////////////////
#ifndef DISPBB_H
#define DISPBB_H
#include "DispBBUI.h"
//---- Start editable code block: headers and declarations

#define      Mode1x1     1
#define      Mode1x2     2
#define      Mode2x1     3
#define      Mode2x2     4
#define      Mode3x3     5
#define      Mode4x4     6
#define      Mode5x5     7
#define      Mode6x6     8
#define      Mode7x7     9
#define      Mode8x8     10
#define      Mode9x9     11
#define      Mode10x10   12

#define      BlowZoom     1
#define      BlowPan      2
#define      BlowDistance 3
#define      BlowAuto     4

//---- End editable code block: headers and declarations


//---- DispBB class declaration

class DispBB : public DispBBUI
{

  public:

    DispBB ( const char *, Widget );
    DispBB ( const char * );
    ~DispBB();
    const char *  className();

    static VkComponent *CreateDispBB( const char *name, Widget parent );

    //---- Start editable code block: DispBB public

    void init();
```

```
} // End RegisterDispBBInterface()


//---- End of generated code

//---- Start editable code block: End of generated code

void DispBB::init()
{
  _mode = Mode1x1;
  _blow = BlowZoom;
}

void DispBB::set_mode(int mode)
{
  _mode = mode;
  switch (mode)
  {
  case Mode1x1: _numH = _numV = 1; break;
  case Mode1x2: _numH = 1; _numV = 2; break;
  case Mode2x1: _numH = 2; _numV = 1; break;
  case Mode2x2: _numH = _numV = 2; break;
  case Mode3x3: _numH = _numV = 3; break;
  case Mode4x4: _numH = _numV = 4; break;
  case Mode5x5: _numH = _numV = 5; break;
  case Mode6x6: _numH = _numV = 6; break;
  case Mode7x7: _numH = _numV = 7; break;
  case Mode8x8: _numH = _numV = 8; break;
  case Mode9x9: _numH = _numV = 9; break;
  case Mode10x10:  _numH = _numV = 10; break;
  default: _numH = _numV = 0; break;
  }

  ((DrawingArea *)_draw) -> set_start_pos();
  ((DrawingArea *)_draw) -> set_layout(_numH, _numV);
  ((DrawingArea *)_draw) -> redisplay();
}

void DispBB::set_blow(int blow)
{
  _blow = blow;
  ((DrawingArea *)_draw) -> set_blow(blow);
}

//---- End editable code block: End of generated code
```

```
////////////////////////////////////////////////////////////////////
// static creation function, for importing class into rapidapp
// or dynamically loading, using VkComponent::loadComponent
////////////////////////////////////////////////////////////////////

VkComponent *DispBB::CreateDispBB( const char *name, Widget parent )
{
    VkComponent *obj =  new DispBB ( name, parent );
    return ( obj );
} // End CreateDispBB

////////////////////////////////////////////////////////////////////
// Function for accessing a description of the dynamic interface
// to this class.
////////////////////////////////////////////////////////////////////

// WARNING: This structure is different than that used with 1.1 RapidApp.
// See the RapidApp release notes for details

struct InterfaceMap {
  char  *resourceName;
  char  *methodName;
  char  *argType;
  char  *definingClass; // Optional, if not this class
  void (VkCallbackObject::*method)(...); // Reserved, do not set
};

void *DispBB::RegisterDispBBInterface()
{
    // This structure registers information about this class
    // that allows RapidApp to create and manipulate an instance.
    // Each entry provides a resource name that will appear in the
    // resource manager palette when an instance of this class is
    // selected, the name of the member function as a string,
    // the type of the single argument to this function, and an.
    // optional argument indicating the class that defines this function.
    // All member functions must have the form
    //
    //     void memberFunction ( Type );
    //
    // where "Type" is one of:
    //    const char *     (Use XmRString)
    //    Boolean          (Use XmRBoolean)
    //    int              (Use XmRInt)
    //    float            (Use XmRFloat)
    //    No argument      (Use VkRNoArg or "NoArg"
    //    A filename       (Use VkRFilename or "Filename")
    //    An enumeration   (Use "Enumeration:ClassName:Type: VALUE1, VALUE2, VALUE3")
    //    A callback       (Use XmRCallback)

    static InterfaceMap map[] = {
    //---- Start editable code block: DispBBUI resource table

      // { "resourceName", "setAttribute", XmRString},
    //---- End editable code block: DispBBUI resource table
      { NULL }, // MUST be NULL terminated
    };

    return map;
```

```
    //::VkUnimplemented ( w, "DispBB::setMode6x6" );
    if(_mode != Mode6x6) set_mode(Mode6x6);

    //---- End editable code block: DispBB setMode6x6
}    // End DispBB::setMode6x6()

void DispBB::setMode7x7 ( Widget w, XtPointer callData )
{
    //---- Start editable code block: DispBB setMode7x7
    XmToggleButtonCallbackStruct *cbs = (XmToggleButtonCallbackStruct*) callData;
    //--- Comment out the following line when DispBB::setMode7x7 is implemented:
    //::VkUnimplemented ( w, "DispBB::setMode7x7" );
    if(_mode != Mode7x7) set_mode(Mode7x7);

    //---- End editable code block: DispBB setMode7x7
}    // End DispBB::setMode7x7()

void DispBB::setMode8x8 ( Widget w, XtPointer callData )
{
    //---- Start editable code block: DispBB setMode8x8
    XmToggleButtonCallbackStruct *cbs = (XmToggleButtonCallbackStruct*) callData;
    //--- Comment out the following line when DispBB::setMode8x8 is implemented:
    //::VkUnimplemented ( w, "DispBB::setMode8x8" );

    if(_mode != Mode8x8) set_mode(Mode8x8);
    //---- End editable code block: DispBB setMode8x8
}    // End DispBB::setMode8x8()

void DispBB::setMode9x9 ( Widget w, XtPointer callData )
{
    //---- Start editable code block: DispBB setMode9x9
    XmToggleButtonCallbackStruct *cbs = (XmToggleButtonCallbackStruct*) callData;
    //--- Comment out the following line when DispBB::setMode9x9 is implemented:
    //::VkUnimplemented ( w, "DispBB::setMode9x9" );
    if(_mode != Mode9x9) set_mode(Mode9x9);

    //---- End editable code block: DispBB setMode9x9
}    // End DispBB::setMode9x9()
```

```
    //---- End editable code block: DispBB setMode2x2

}     // End DispBB::setMo    x2()


void DispBB::setMode3x3 ( Widget w, XtPointer callData )
{
    //---- Start editable code block: DispBB setMode3x3

    XmToggleButtonCallbackStruct *cbs = (XmToggleButtonCallbackStruct*) callData;

    //--- Comment out the following line when DispBB::setMode3x3 is implemented:

    //::VkUnimplemented ( w, "DispBB::setMode3x3" );
    if(_mode != Mode3x3) set_mode(Mode3x3);



    //---- End editable code block: DispBB setMode3x3

}     // End DispBB::setMode3x3()


void DispBB::setMode4x4 ( Widget w, XtPointer callData )
{
    //---- Start editable code block: DispBB setMode4x4

    XmToggleButtonCallbackStruct *cbs = (XmToggleButtonCallbackStruct*) callData;

    //--- Comment out the following line when DispBB::setMode4x4 is implemented:

    //::VkUnimplemented ( w, "DispBB::setMode4x4" );

    if(_mode != Mode4x4) set_mode(Mode4x4);


    //---- End editable code block: DispBB setMode4x4

}     // End DispBB::setMode4x4()


void DispBB::setMode5x5 ( Widget w, XtPointer callData )
{
    //---- Start editable code block: DispBB setMode5x5

    XmToggleButtonCallbackStruct *cbs = (XmToggleButtonCallbackStruct*) callData;

    //--- Comment out the following line when DispBB::setMode5x5 is implemented:

    //::VkUnimplemented ( w, "DispBB::setMode5x5" );

    if(_mode != Mode5x5) set_mode(Mode5x5);


    //---- End editable code block: DispBB setMode5x5

}     // End DispBB::setMode5x5()


void DispBB::setMode6x6 ( Widget w, XtPointer callData )
{
    //---- Start editable code block: DispBB setMode6x6

    XmToggleButtonCallbackStruct *cbs = (XmToggleButtonCallbackStruct*) callData;

    //--- Comment out the following line when DispBB::setMode6x6 is implemented:
```

```
void DispBB::setMode1x1 ( Widget w, XtPointer callData )
{
    //---- Start editable code block: DispBB setMode1x1

    XmToggleButtonCallbackStruct *cbs = (XmToggleButtonCallbackStruct*) callData;

    //--- Comment out the following line when DispBB::setMode1x1 is implemented:

    //::VkUnimplemented ( w, "DispBB::setMode1x1" );
    if(_mode != Mode1x1) set_mode(Mode1x1);


    //---- End editable code block: DispBB setMode1x1

}    // End DispBB::setMode1x1()


void DispBB::setMode1x2 ( Widget w, XtPointer callData )
{
    //---- Start editable code block: DispBB setMode1x2

    XmToggleButtonCallbackStruct *cbs = (XmToggleButtonCallbackStruct*) callData;

    //--- Comment out the following line when DispBB::setMode1x2 is implemented:

    //::VkUnimplemented ( w, "DispBB::setMode1x2" );
    if(_mode != Mode1x2) set_mode(Mode1x2);



    //---- End editable code block: DispBB setMode1x2

}    // End DispBB::setMode1x2()


void DispBB::setMode2x1 ( Widget w, XtPointer callData )
{
    //---- Start editable code block: DispBB setMode2x1

    XmToggleButtonCallbackStruct *cbs = (XmToggleButtonCallbackStruct*) callData;

    //--- Comment out the following line when DispBB::setMode2x1 is implemented:

    //::VkUnimplemented ( w, "DispBB::setMode2x1" );
    if(_mode != Mode2x1) set_mode(Mode2x1);



    //---- End editable code block: DispBB setMode2x1

}    // End DispBB::setMode2x1()


void DispBB::setMode2x2 ( Widget w, XtPointer callData )
{
    //---- Start editable code block: DispBB setMode2x2

    XmToggleButtonCallbackStruct *cbs = (XmToggleButtonCallbackStruct*) callData;

    //--- Comment out the following line when DispBB::setMode2x2 is implemented:

    //::VkUnimplemented ( w, "DispBB::setMode2x2" );

    if(_mode != Mode2x2) set_mode(Mode2x2);
```

```
    //::VkUnimplemented ( w, "DispBB::blowUpZoom" );
    if(_blow != BlowZoom) set_blow(BlowZoom);

    //---- End editable code block: DispBB blowUpZoom
}    // End DispBB::blowUpZoom()

void DispBB::magicDown ( Widget w, XtPointer callData )
{
    //---- Start editable code block: DispBB magicDown
    XmArrowButtonCallbackStruct *cbs = (XmArrowButtonCallbackStruct*) callData;
    //--- Comment out the following line when DispBB::magicDown is implemented:
    //::VkUnimplemented ( w, "DispBB::magicDown" );
    ((DrawingArea *)_draw) -> set_start_pos(1);

    //---- End editable code block: DispBB magicDown
}    // End DispBB::magicDown()

void DispBB::magicUp ( Widget w, XtPointer callData )
{
    //---- Start editable code block: DispBB magicUp
    XmArrowButtonCallbackStruct *cbs = (XmArrowButtonCallbackStruct*) callData;
    //--- Comment out the following line when DispBB::magicUp is implemented:
    //::VkUnimplemented ( w, "DispBB::magicUp" );
    ((DrawingArea *)_draw) -> set_start_pos(-1);


    //---- End editable code block: DispBB magicUp
}    // End DispBB::magicUp()

void DispBB::setMode10x10 ( Widget w, XtPointer callData )
{
    //---- Start editable code block: DispBB setMode10x10
    XmToggleButtonCallbackStruct *cbs = (XmToggleButtonCallbackStruct*) callData;
    //--- Comment out the following line when DispBB::setMode10x10 is implemented:
    //::VkUnimplemented ( w, "DispBB::setMode10x10" );
    if(_mode != Mode10x10) set_mode(Mode10x10);


    //---- End editable code block: DispBB setMode10x10
}    // End DispBB::setMode10x10()
```

```
void DispBB::blowUpButton( Widget w, XtPointer callData )
{
    //---- Start editable code block: DispBB blowUpButton

    XmPushButtonCallbackStruct *cbs = (XmPushButtonCallbackStruct*) callData;

    //--- Comment out the following line when DispBB::blowUpButton is implemented:

    //::VkUnimplemented ( w, "DispBB::blowUpButton" );

    if(_mode != Mode1x1)
    {
      set_mode(Mode1x1);
    }

    //---- End editable code block: DispBB blowUpButton

}    // End DispBB::blowUpButton()

void DispBB::blowUpDistance ( Widget w, XtPointer callData )
{
    //---- Start editable code block: DispBB blowUpDistance

    XmToggleButtonCallbackStruct *cbs = (XmToggleButtonCallbackStruct*) callData;

    //--- Comment out the following line when DispBB::blowUpDistance is implemented:

    //::VkUnimplemented ( w, "DispBB::blowUpDistance" );

    if(_blow != BlowDistance) set_blow(BlowDistance);

    //---- End editable code block: DispBB blowUpDistance

}    // End DispBB::blowUpDistance()

void DispBB::blowUpPan ( Widget w, XtPointer callData )
{
    //---- Start editable code block: DispBB blowUpPan

    XmToggleButtonCallbackStruct *cbs = (XmToggleButtonCallbackStruct*) callData;

    //--- Comment out the following line when DispBB::blowUpPan is implemented:

    //::VkUnimplemented ( w, "DispBB::blowUpPan" );

    if(_blow != BlowPan) set_blow(BlowPan);

    //---- End editable code block: DispBB blowUpPan

}    // End DispBB::blowUpPan()

void DispBB::blowUpZoom ( Widget w, XtPointer callData )
{
    //---- Start editable code block: DispBB blowUpZoom

    XmToggleButtonCallbackStruct *cbs = (XmToggleButtonCallbackStruct*) callData;

    //--- Comment out the following line when DispBB::blowUpZoom is implemented:
```

```
//---- End editable code block: headers and declarations


//---- DispBB Constructor

DispBB::DispBB(const char *name, Widget parent) :
                    DispBBUI(name, parent)
{
    // This constructor calls DispBBUI(parent, name)
    // which calls DispBBUI::create() to create
    // the widgets for this component. Any code added here
    // is called after the component's interface has been built

    //---- Start editable code block: DispBB constructor

    init();

    //---- End editable code block: DispBB constructor



}    // End Constructor



DispBB::DispBB(const char *name) :
                    DispBBUI(name)
 {
    // This constructor calls DispBBUI(name)
    // which does not create any widgets. Usually, this
    // constructor is not used

    //---- Start editable code block: DispBB constructor 2

    init();

    //---- End editable code block: DispBB constructor 2



}    // End Constructor



DispBB::~DispBB()
{
    // The base class destructors are responsible for
    // destroying all widgets and objects used in this component.
    // Only additional items created directly in this class
    // need to be freed here.

    //---- Start editable code block: DispBB destructor


    //---- End editable code block: DispBB destructor



}    // End Destructor


const char * DispBB::className() // classname
{
    return ("DispBB");
} // End className()
```

```
////////////////////////////////////////////////////////////
//
// Source file for DispBB
//
//    This file is generated by RapidApp 1.2
//
//    This class is derived from DispBBUI which
//    implements the user interface created in
//    RapidApp. This class contains virtual
//    functions that are called from the user interface.
//
//    When you modify this source, limit your changes to
//    modifying the sections between the
//    "//---- Start/End editable code block" markers
//
//    This will allow RapidApp to integrate changes more easily
//
//    This class is a ViewKit user interface "component".
//    For more information on how components are used, see the
//    "ViewKit Programmers' Manual", and the RapidApp
//    User's Guide.
////////////////////////////////////////////////////////////

#include "DispBB.h"
#include <Vk/VkEZ.h>
#include <Xm/ArrowB.h>
#include <Xm/BulletinB.h>
#include <Xm/PushB.h>
#include <Xm/RowColumn.h>
#include <Xm/ToggleB.h>
#include <Vk/VkResource.h>


extern void VkUnimplemented ( Widget, const char * );

////////////////////////////////////////////////////////////////////////////
// The following non-container elements are created by DispBBUI and are
// available as protected data members inherited by this class
//
//  XmToggleButton                  _mode1x1
//  XmToggleButton                  _mode1x2
//  XmToggleButton                  _mode2x1
//  XmToggleButton                  _mode2x2
//  XmToggleButton                  _mode3x3
//  XmToggleButton                  _mode4x4
//  XmToggleButton                  _mode5x5
//  XmToggleButton                  _mode6x6
//  XmToggleButton                  _mode7x7
//  XmToggleButton                  _mode8x8
//  XmToggleButton                  _mode9x9
//  XmToggleButton                  _mode10x10
//  XmArrowButton                   _arrowUp
//  XmArrowButton                   _arrowDown
//  XmPushButton                    _blowButton
//  XmToggleButton                  _blowZoom
//  XmToggleButton                  _blowPan
//  XmToggleButton                  _blowDistance
//
////////////////////////////////////////////////////////////////////////////


//---- Start editable code block: headers and declarations

#include "DrawingArea.h"
```

```
    static void setMode6x6Callback ( Widget, XtPointer, XtPointer );
    static void setMode7x[illegible]allback ( Widget, XtPointer, X[illegible]inter );
    static void setMode8x[illegible]allback ( Widget, XtPointer, X[illegible]inter );
    static void setMode9x9Callback ( Widget, XtPointer, XtPointer );

    //---- Start editable code block: DispBB private


    //---- End editable code block: DispBB private
};
//---- Start editable code block: End of generated code


//---- End editable code block: End of generated code

#endif
```

```
    Widget  _mode10x10;
    Widget  _mode1x1;
    Widget  _mode1x2;
    Widget  _mode2x1;
    Widget  _mode2x2;
    Widget  _mode3x3;
    Widget  _mode4x4;
    Widget  _mode5x5;
    Widget  _mode6x6;
    Widget  _mode7x7;
    Widget  _mode8x8;
    Widget  _mode9x9;
    Widget  _radioboxBlowUp;


    // These virtual functions are called from the private callbacks (below)
    // Intended to be overriden in derived classes to define actions

    virtual void blowUpButton ( Widget, XtPointer );
    virtual void blowUpDistance ( Widget, XtPointer );
    virtual void blowUpPan ( Widget, XtPointer );
    virtual void blowUpZoom ( Widget, XtPointer );
    virtual void magicDown ( Widget, XtPointer );
    virtual void magicUp ( Widget, XtPointer );
    virtual void setMode10x10 ( Widget, XtPointer );
    virtual void setMode1x1 ( Widget, XtPointer );
    virtual void setMode1x2 ( Widget, XtPointer );
    virtual void setMode2x1 ( Widget, XtPointer );
    virtual void setMode2x2 ( Widget, XtPointer );
    virtual void setMode3x3 ( Widget, XtPointer );
    virtual void setMode4x4 ( Widget, XtPointer );
    virtual void setMode5x5 ( Widget, XtPointer );
    virtual void setMode6x6 ( Widget, XtPointer );
    virtual void setMode7x7 ( Widget, XtPointer );
    virtual void setMode8x8 ( Widget, XtPointer );
    virtual void setMode9x9 ( Widget, XtPointer );

    //---- Start editable code block: DispBB protected


    //---- End editable code block: DispBB protected


private:

    // Array of default resources

    static String      _defaultDispBBUIResources[];


    // Callbacks to interface with Motif

    static void blowUpButtonCallback ( Widget, XtPointer, XtPointer );
    static void blowUpDistanceCallback ( Widget, XtPointer, XtPointer );
    static void blowUpPanCallback ( Widget, XtPointer, XtPointer );
    static void blowUpZoomCallback ( Widget, XtPointer, XtPointer );
    static void magicDownCallback ( Widget, XtPointer, XtPointer );
    static void magicUpCallback ( Widget, XtPointer, XtPointer );
    static void setMode10x10Callback ( Widget, XtPointer, XtPointer );
    static void setMode1x1Callback ( Widget, XtPointer, XtPointer );
    static void setMode1x2Callback ( Widget, XtPointer, XtPointer );
    static void setMode2x1Callback ( Widget, XtPointer, XtPointer );
    static void setMode2x2Callback ( Widget, XtPointer, XtPointer );
    static void setMode3x3Callback ( Widget, XtPointer, XtPointer );
    static void setMode4x4Callback ( Widget, XtPointer, XtPointer );
    static void setMode5x5Callback ( Widget, XtPointer, XtPointer );
```

```
////////////////////////////////////////////////////////////
//
// Header file for DispBBUI
//
//    This file is generated by RapidApp 1.2
//
//    This class implements the user interface portion of a class
//    Normally it is not used directly.
//    Instead the subclass, DispBB is instantiated
//
//    To extend or alter the behavior of this class, you should
//    modify the DispBB files
//
//    Restrict changes to those sections between
//    the "//--- Start/End editable code block" markers
//
//    This will allow RapidApp to integrate changes more easily
//
//    This class is a ViewKit user interface "component".
//    For more information on how components are used, see the
//    "ViewKit Programmers' Manual", and the RapidApp
//    User's Guide.
//
////////////////////////////////////////////////////////////
#ifndef DISPBBUI_H
#define DISPBBUI_H
#include <Vk/VkComponent.h>


//---- Start editable code block: headers and declarations


//---- End editable code block: headers and declarations


class DispBBUI : public VkComponent
{

  public:

    DispBBUI ( const char *, Widget );
    DispBBUI ( const char * );
    ~DispBBUI();
    void create ( Widget );
    const char *  className();

    //---- Start editable code block: DispBB public


    //---- End editable code block: DispBB public


  protected:


    // Widgets created by this class

    Widget  _arrowDown;
    Widget  _arrowUp;
    Widget  _blowButton;
    Widget  _blowDistance;
    Widget  _blowPan;
    Widget  _blowZoom;
    Widget  _dispBB;
    Widget  _magicModeRB;
```

```
    // This virtual function is called from setMode4x4Callback.
    // This function is normally overriden by a derived class.

}

void DispBBUI::setMode5x5 ( Widget, XtPointer )
{
    // This virtual function is called from setMode5x5Callback.
    // This function is normally overriden by a derived class.

}

void DispBBUI::setMode6x6 ( Widget, XtPointer )
{
    // This virtual function is called from setMode6x6Callback.
    // This function is normally overriden by a derived class.

}

void DispBBUI::setMode7x7 ( Widget, XtPointer )
{
    // This virtual function is called from setMode7x7Callback.
    // This function is normally overriden by a derived class.

}

void DispBBUI::setMode8x8 ( Widget, XtPointer )
{
    // This virtual function is called from setMode8x8Callback.
    // This function is normally overriden by a derived class.

}

void DispBBUI::setMode9x9 ( Widget, XtPointer )
{
    // This virtual function is called from setMode9x9Callback.
    // This function is normally overriden by a derived class.

}


//---- Start editable code block: End of generated code


//---- End editable code block: End of generated code
```

```
void DispBBUI::blowUpZoom( Widget, XtPointer )
{
    // This virtual function is called from blowUpZoomCallback.
    // This function is normally overriden by a derived class.

}

void DispBBUI::magicDown ( Widget, XtPointer )
{
    // This virtual function is called from magicDownCallback.
    // This function is normally overriden by a derived class.

}

void DispBBUI::magicUp ( Widget, XtPointer )
{
    // This virtual function is called from magicUpCallback.
    // This function is normally overriden by a derived class.

}

void DispBBUI::setMode10x10 ( Widget, XtPointer )
{
    // This virtual function is called from setMode10x10Callback.
    // This function is normally overriden by a derived class.

}

void DispBBUI::setMode1x1 ( Widget, XtPointer )
{
    // This virtual function is called from setMode1x1Callback.
    // This function is normally overriden by a derived class.

}

void DispBBUI::setMode1x2 ( Widget, XtPointer )
{
    // This virtual function is called from setMode1x2Callback.
    // This function is normally overriden by a derived class.

}

void DispBBUI::setMode2x1 ( Widget, XtPointer )
{
    // This virtual function is called from setMode2x1Callback.
    // This function is normally overriden by a derived class.

}

void DispBBUI::setMode2x2 ( Widget, XtPointer )
{
    // This virtual function is called from setMode2x2Callback.
    // This function is normally overriden by a derived class.

}

void DispBBUI::setMode3x3 ( Widget, XtPointer )
{
    // This virtual function is called from setMode3x3Callback.
    // This function is normally overriden by a derived class.

}

void DispBBUI::setMode4x4 ( Widget, XtPointer )
{
```

```
                                    XtPointer clientData,
                                    XtPointer callData )
{
    DispBBUI* obj = ( DispBBUI * ) clientData;
    obj->setMode5x5 ( w, callData );
}

void DispBBUI::setMode6x6Callback ( Widget    w,
                                    XtPointer clientData,
                                    XtPointer callData )
{
    DispBBUI* obj = ( DispBBUI * ) clientData;
    obj->setMode6x6 ( w, callData );
}

void DispBBUI::setMode7x7Callback ( Widget    w,
                                    XtPointer clientData,
                                    XtPointer callData )
{
    DispBBUI* obj = ( DispBBUI * ) clientData;
    obj->setMode7x7 ( w, callData );
}

void DispBBUI::setMode8x8Callback ( Widget    w,
                                    XtPointer clientData,
                                    XtPointer callData )
{
    DispBBUI* obj = ( DispBBUI * ) clientData;
    obj->setMode8x8 ( w, callData );
}

void DispBBUI::setMode9x9Callback ( Widget    w,
                                    XtPointer clientData,
                                    XtPointer callData )
{
    DispBBUI* obj = ( DispBBUI * ) clientData;
    obj->setMode9x9 ( w, callData );
}



////////////////////////////////////////////////////////////////
// The following functions are called from the menu items
// in this window.
////////////////////////////////////

void DispBBUI::blowUpButton ( Widget, XtPointer )
{
    // This virtual function is called from blowUpButtonCallback.
    // This function is normally overriden by a derived class.

}

void DispBBUI::blowUpDistance ( Widget, XtPointer )
{
    // This virtual function is called from blowUpDistanceCallback.
    // This function is normally overriden by a derived class.

}

void DispBBUI::blowUpPan ( Widget, XtPointer )
{
    // This virtual function is called from blowUpPanCallback.
    // This function is normally overriden by a derived class.

}
```

```
void DispBBUI::magicUpCallback ( Widget    w,
                                 XtPointer clientData,
                                 XtPointer callData )
{
    DispBBUI* obj = ( DispBBUI * ) clientData;
    obj->magicUp ( w, callData );
}

void DispBBUI::setMode10x10Callback ( Widget    w,
                                      XtPointer clientData,
                                      XtPointer callData )
{
    DispBBUI* obj = ( DispBBUI * ) clientData;
    obj->setMode10x10 ( w, callData );
}

void DispBBUI::setMode1x1Callback ( Widget    w,
                                    XtPointer clientData,
                                    XtPointer callData )
{
    DispBBUI* obj = ( DispBBUI * ) clientData;
    obj->setMode1x1 ( w, callData );
}

void DispBBUI::setMode1x2Callback ( Widget    w,
                                    XtPointer clientData,
                                    XtPointer callData )
{
    DispBBUI* obj = ( DispBBUI * ) clientData;
    obj->setMode1x2 ( w, callData );
}

void DispBBUI::setMode2x1Callback ( Widget    w,
                                    XtPointer clientData,
                                    XtPointer callData )
{
    DispBBUI* obj = ( DispBBUI * ) clientData;
    obj->setMode2x1 ( w, callData );
}

void DispBBUI::setMode2x2Callback ( Widget    w,
                                    XtPointer clientData,
                                    XtPointer callData )
{
    DispBBUI* obj = ( DispBBUI * ) clientData;
    obj->setMode2x2 ( w, callData );
}

void DispBBUI::setMode3x3Callback ( Widget    w,
                                    XtPointer clientData,
                                    XtPointer callData )
{
    DispBBUI* obj = ( DispBBUI * ) clientData;
    obj->setMode3x3 ( w, callData );
}

void DispBBUI::setMode4x4Callback ( Widget    w,
                                    XtPointer clientData,
                                    XtPointer callData )
{
    DispBBUI* obj = ( DispBBUI * ) clientData;
    obj->setMode4x4 ( w, callData );
}

void DispBBUI::setMode5x5Callback ( Widget    w,
```

```
    XtAddCallback ( _blowUpDistance,
                    XmNvalueChangedCallback,
                    &DispBBUI::blowUpDistanceCallback,
                    (XtPointer) this );


    //---- Start editable code block: DispBBUI create


    //---- End editable code block: DispBBUI create
}

const char * DispBBUI::className()
{
    return ("DispBBUI");
}    // End className()



////////////////////////////////////////////////////////////////
// The following functions are static member functions used to
// interface with Motif.
////////////////////////////////////

void DispBBUI::blowUpButtonCallback ( Widget    w,
                                      XtPointer clientData,
                                      XtPointer callData )
{
    DispBBUI* obj = ( DispBBUI * ) clientData;
    obj->blowUpButton ( w, callData );
}

void DispBBUI::blowUpDistanceCallback ( Widget    w,
                                        XtPointer clientData,
                                        XtPointer callData )
{
    DispBBUI* obj = ( DispBBUI * ) clientData;
    obj->blowUpDistance ( w, callData );
}

void DispBBUI::blowUpPanCallback ( Widget    w,
                                   XtPointer clientData,
                                   XtPointer callData )
{
    DispBBUI* obj = ( DispBBUI * ) clientData;
    obj->blowUpPan ( w, callData );
}

void DispBBUI::blowUpZoomCallback ( Widget    w,
                                    XtPointer clientData,
                                    XtPointer callData )
{
    DispBBUI* obj = ( DispBBUI * ) clientData;
    obj->blowUpZoom ( w, callData );
}

void DispBBUI::magicDownCallback ( Widget    w,
                                   XtPointer clientData,
                                   XtPointer callData )
{
    DispBBUI* obj = ( DispBBUI * ) clientData;
    obj->magicDown ( w, callData );
}
```

```
                                        (XtPointer) NULL );

XtAddCallback ( _arrowDown,
                XmNactivateCallback,
                &DispBBUI::magicDownCallback,
                (XtPointer) this );


_blowButton = XtVaCreateManagedWidget  ( "blowButton",
                                         xmPushButtonWidgetClass,
                                         _baseWidget,
                                         XmNlabelType, XmSTRING,
                                         XmNx, 171,
                                         XmNy, 150,
                                         XmNwidth, 60,
                                         XmNheight, 60,
                                         (XtPointer) NULL );

XtAddCallback ( _blowButton,
                XmNactivateCallback,
                &DispBBUI::blowUpButtonCallback,
                (XtPointer) this );


_radioboxBlowUp = XtVaCreateManagedWidget  ( "radioboxBlowUp",
                                             xmRowColumnWidgetClass,
                                             _baseWidget,
                                             XmNpacking, XmPACK_COLUMN,
                                             XmNradioBehavior, True,
                                             XmNradioAlwaysOne, True,
                                             XmNx, 292,
                                             XmNy, 132,
                                             XmNwidth, 88,
                                             XmNheight, 90,
                                             (XtPointer) NULL );


_blowZoom = XtVaCreateManagedWidget  ( "blowZoom",
                                       xmToggleButtonWidgetClass,
                                       _radioboxBlowUp,
                                       XmNlabelType, XmSTRING,
                                       (XtPointer) NULL );

XtAddCallback ( _blowZoom,
                XmNvalueChangedCallback,
                &DispBBUI::blowUpZoomCallback,
                (XtPointer) this );


_blowPan = XtVaCreateManagedWidget  ( "blowPan",
                                      xmToggleButtonWidgetClass,
                                      _radioboxBlowUp,
                                      XmNlabelType, XmSTRING,
                                      (XtPointer) NULL );

XtAddCallback ( _blowPan,
                XmNvalueChangedCallback,
                &DispBBUI::blowUpPanCallback,
                (XtPointer) this );


_blowDistance = XtVaCreateManagedWidget  ( "blowDistance",
                                           xmToggleButtonWidgetClass,
                                           _radioboxBlowUp,
                                           XmNlabelType, XmSTRING,
                                           (XtPointer) NULL );
```

```
XtAddCallback ( _mode[redacted],
                XmNva[redacted]ChangedCallback,
                &DispBBUI::setMode7x7Callback,
                (XtPointer) this );


_mode8x8 = XtVaCreateManagedWidget  ( "mode8x8",
                                      xmToggleButtonWidgetClass,
                                      _magicModeRB,
                                      XmNlabelType, XmSTRING,
                                      (XtPointer) NULL );

XtAddCallback ( _mode8x8,
                XmNvalueChangedCallback,
                &DispBBUI::setMode8x8Callback,
                (XtPointer) this );


_mode9x9 = XtVaCreateManagedWidget  ( "mode9x9",
                                      xmToggleButtonWidgetClass,
                                      _magicModeRB,
                                      XmNlabelType, XmSTRING,
                                      (XtPointer) NULL );

XtAddCallback ( _mode9x9,
                XmNvalueChangedCallback,
                &DispBBUI::setMode9x9Callback,
                (XtPointer) this );


_mode10x10 = XtVaCreateManagedWidget  ( "mode10x10",
                                        xmToggleButtonWidgetClass,
                                        _magicModeRB,
                                        XmNlabelType, XmSTRING,
                                        (XtPointer) NULL );

XtAddCallback ( _mode10x10,
                XmNvalueChangedCallback,
                &DispBBUI::setMode10x10Callback,
                (XtPointer) this );


_arrowUp = XtVaCreateManagedWidget  ( "arrowUp",
                                      xmArrowButtonWidgetClass,
                                      _baseWidget,
                                      XmNx, 170,
                                      XmNy, 10,
                                      XmNwidth, 60,
                                      XmNheight, 60,
                                      (XtPointer) NULL );

XtAddCallback ( _arrowUp,
                XmNactivateCallback,
                &DispBBUI::magicUpCallback,
                (XtPointer) this );


_arrowDown = XtVaCreateManagedWidget  ( "arrowDown",
                                        xmArrowButtonWidgetClass,
                                        _baseWidget,
                                        XmNarrowDirection, XmARROW_DOWN,
                                        XmNx, 170,
                                        XmNy, 290,
                                        XmNwidth, 60,
                                        XmNheight, 60,
```

```
_mode2x2 = XtVaCreateManagedWidget  ( "mode2x2",
                                      xmToggleButtonWidgetClass,
                                      _magicModeRB,
                                      XmNlabelType, XmSTRING,
                                      (XtPointer) NULL );

XtAddCallback ( _mode2x2,
                XmNvalueChangedCallback,
                &DispBBUI::setMode2x2Callback,
                (XtPointer) this );


_mode3x3 = XtVaCreateManagedWidget  ( "mode3x3",
                                      xmToggleButtonWidgetClass,
                                      _magicModeRB,
                                      XmNlabelType, XmSTRING,
                                      (XtPointer) NULL );

XtAddCallback ( _mode3x3,
                XmNvalueChangedCallback,
                &DispBBUI::setMode3x3Callback,
                (XtPointer) this );


_mode4x4 = XtVaCreateManagedWidget  ( "mode4x4",
                                      xmToggleButtonWidgetClass,
                                      _magicModeRB,
                                      XmNlabelType, XmSTRING,
                                      (XtPointer) NULL );

XtAddCallback ( _mode4x4,
                XmNvalueChangedCallback,
                &DispBBUI::setMode4x4Callback,
                (XtPointer) this );


_mode5x5 = XtVaCreateManagedWidget  ( "mode5x5",
                                      xmToggleButtonWidgetClass,
                                      _magicModeRB,
                                      XmNlabelType, XmSTRING,
                                      (XtPointer) NULL );

XtAddCallback ( _mode5x5,
                XmNvalueChangedCallback,
                &DispBBUI::setMode5x5Callback,
                (XtPointer) this );


_mode6x6 = XtVaCreateManagedWidget  ( "mode6x6",
                                      xmToggleButtonWidgetClass,
                                      _magicModeRB,
                                      XmNlabelType, XmSTRING,
                                      (XtPointer) NULL );

XtAddCallback ( _mode6x6,
                XmNvalueChangedCallback,
                &DispBBUI::setMode6x6Callback,
                (XtPointer) this );


_mode7x7 = XtVaCreateManagedWidget  ( "mode7x7",
                                      xmToggleButtonWidgetClass,
                                      _magicModeRB,
                                      XmNlabelType, XmSTRING,
                                      (XtPointer) NULL );
```

```
// Create an unmanaged widget as the top of the widget hierarchy

_baseWidget = _dispBB = XtVaCreateWidget ( _name,
                                           xmBulletinBoardWidgetClass,
                                           parent,
                                           XmNresizePolicy, XmRESIZE_GROW,
                                           (XtPointer) NULL );

// install a callback to guard against unexpected widget destruction

installDestroyHandler();


// Create widgets used in this component
// All variables are data members of this class

_magicModeRB = XtVaCreateManagedWidget  ( "magicModeRB",
                                          xmRowColumnWidgetClass,
                                          _baseWidget,
                                          XmNpacking, XmPACK_COLUMN,
                                          XmNradioBehavior, True,
                                          XmNradioAlwaysOne, True,
                                          XmNx, 32,
                                          XmNy, 10,
                                          XmNwidth, 80,
                                          XmNheight, 351,
                                          (XtPointer) NULL );


_mode1x1 = XtVaCreateManagedWidget  ( "mode1x1",
                                      xmToggleButtonWidgetClass,
                                      _magicModeRB,
                                      XmNlabelType, XmSTRING,
                                      (XtPointer) NULL );

XtAddCallback ( _mode1x1,
                XmNvalueChangedCallback,
                &DispBBUI::setMode1x1Callback,
                (XtPointer) this );


_mode1x2 = XtVaCreateManagedWidget  ( "mode1x2",
                                      xmToggleButtonWidgetClass,
                                      _magicModeRB,
                                      XmNlabelType, XmSTRING,
                                      (XtPointer) NULL );

XtAddCallback ( _mode1x2,
                XmNvalueChangedCallback,
                &DispBBUI::setMode1x2Callback,
                (XtPointer) this );


_mode2x1 = XtVaCreateManagedWidget  ( "mode2x1",
                                      xmToggleButtonWidgetClass,
                                      _magicModeRB,
                                      XmNlabelType, XmSTRING,
                                      (XtPointer) NULL );

XtAddCallback ( _mode2x1,
                XmNvalueChangedCallback,
                &DispBBUI::setMode2x1Callback,
                (XtPointer) this );
```

```
        (char*)NULL
};

DispBBUI::DispBBUI ( const char *name ) : VkComponent ( name )
{
    // No widgets are created by this constructor.
    // If an application creates a component using this constructor,
    // It must explictly call create at a later time.
    // This is mostly useful when adding pre-widget creation
    // code to a derived class constructor.

    //---- Start editable code block: DispBB constructor 2


    //---- End editable code block: DispBB constructor 2


}    // End Constructor



DispBBUI::DispBBUI ( const char *name, Widget parent ) : VkComponent ( name )
{
    //---- Start editable code block: DispBB pre-create


    //---- End editable code block: DispBB pre-create



    // Call creation function to build the widget tree.

     create ( parent );

    //---- Start editable code block: DispBB constructor


    //---- End editable code block: DispBB constructor


}    // End Constructor


DispBBUI::~DispBBUI()
{
    // Base class destroys widgets

    //---- Start editable code block: DispBBUI destructor


    //---- End editable code block: DispBBUI destructor
}    // End destructor



void DispBBUI::create ( Widget parent )
{
    Arg      args[7];
    Cardinal count;
    count = 0;

    // Load any class-defaulted resources for this object

    setDefaultResources ( parent, _defaultDispBBUIResources  );
```

```
////////////////////////////////////////////////////////////
//
// Source file for DispBBUI
//
//    This class implements the user interface created in
//    RapidApp.
//
//    Restrict changes to those sections between
//    the "//--- Start/End editable code block" markers
//
//    This will allow RapidApp to integrate changes more easily
//
//    This class is a ViewKit user interface "component".
//    For more information on how components are used, see the
//    "ViewKit Programmers' Manual", and the RapidApp
//    User's Guide.
//
//
////////////////////////////////////////////////////////////


#include "DispBBUI.h" // Generated header file for this class

#include <Xm/ArrowB.h>
#include <Xm/BulletinB.h>
#include <Xm/PushB.h>
#include <Xm/RowColumn.h>
#include <Xm/ToggleB.h>
#include <Vk/VkResource.h>
//---- Start editable code block: headers and declarations


//---- End editable code block: headers and declarations



// These are default resources for widgets in objects of this class
// All resources will be prepended by *<name> at instantiation,
// where <name> is the name of the specific instance, as well as the
// name of the baseWidget. These are only defaults, and may be overriden
// in a resource file by providing a more specific resource name

String  DispBBUI::_defaultDispBBUIResources[] = {
        "*blowButton.labelString:  GO",
        "*blowDistance.labelString:  Distance",
        "*blowPan.labelString:  Pan",
        "*blowZoom.labelString:  Zoom",
        "*mode10x10.labelString:  10 x 10",
        "*mode1x1.labelString:  1 x 1",
        "*mode1x2.labelString:  1 x 2",
        "*mode2x1.labelString:  2 x 1",
        "*mode2x2.labelString:  2 x 2",
        "*mode3x3.labelString:  3 x 3",
        "*mode4x4.labelString:  4 x 4",
        "*mode5x5.labelString:  5 x 5",
        "*mode6x6.labelString:  6 x 6",
        "*mode7x7.labelString:  7 x 7",
        "*mode8x8.labelString:  8 x 8",
        "*mode9x9.labelString:  9 x 9",
        "*tabLabel:  Display",

        //---- Start editable code block: DispBBUI Default Resources


        //---- End editable code block: DispBBUI Default Resources
```

```
    {
      char str[50];
      sprintf(str, "%7.3f"[illegible]);
      XmTextFieldSetString(_textfieldPixels, str);
      XmTextFieldSetString(_magicDeck -> get_textfieldImagePixel(), str);

      float pixsize = atof(XmTextFieldGetString(_magicDeck -> get_textfieldPixsize()));
      sprintf(str, "%7.3f", d*pixsize);
      XmTextFieldSetString(_textfieldDistance, str);
      XmTextFieldSetString(_magicDeck -> get_textfieldImageDistance(), str);
    }
}

void MagicBB::anatomy_info(char *a)
{
   printf("anatomy_info :  %s\n", a);
   if(a != NULL)
   {
     CanvasString *cs = new CanvasString(a);
     cs -> trim(' ');
     cs -> trim('\n');
     cs -> trim(' ');
     printf("anatomy_info :::  %s\n", cs -> get());
     XmTextFieldSetString(_textfieldAnatomy, cs -> get());
   }
}

void MagicBB::set_measured(float d)
{
     printf(" 11 CANVAS = %d\n", _whichVessel);
    ((ModelDraw *)_modelDrawArea) -> set_measured(_whichVessel, d);
}

void MagicBB::set_blow(int blow)
{
  _blow = blow;
  ((DrawingArea *)_draw) -> set_blow(blow);
}

//---- End editable code block: End of generated code
```

```
        x++;
      } while( files -> g[redacted]Next() && x <= 160 ); // && x[redacted] 3);
    }
      x = 0;
      do
      {
        aFile = files -> getCursor();
        strcpy(cmd, dir -> get());
        strcat(cmd, aFile.get());
        if(mimg[x].open_file(flag, cmd))
        {
          printf("  %d   read_file succeed!  %s\n", x, cmd);
          mimg[x].init_grayImg();
          if(flag == 0)
            mimg[x].update_grayImg(0.0, 256.0);
          else if(flag == 1)
            mimg[x].update_grayImg(300.0, 1024.0);
          else if(flag == 2)
          {
            if(aFile.equal("cd.001"))
              mimg[x].update_grayImg(0, 0);
            else if(aFile.equal("mag.001"))
              mimg[x].update_grayImg(0, 0);
            else if(aFile.equal("pd.001"))
              mimg[x].update_grayImg(0, 0);
            else if(aFile.equal("perpcd.001"))
              mimg[x].update_grayImg(0, 0);
          }
          x++;
        }
      } while( files -> gotoNext() && x <= 20); // && x <= 3);
    */

    delete aux2;

    ((DrawingArea *)_draw) -> set(0, x, mimg);


    MODEL_WIDTH = 723;
    MODEL_HEIGHT = 964;

    _modelDrawArea = new ModelDraw(0, 0, MODEL_WIDTH,MODEL_HEIGHT, "sim", _scrolledMode
    _modelDrawArea -> show();
    _whichVessel = -1;
    ((ModelDraw *)_modelDrawArea) -> set(this);
}

void MagicBB::vessel_info(int vessel, float d)
{
   _whichVessel = vessel;
   printf("00 CANVAS = %d\n", _whichVessel);
   if(_whichVessel >= 0)
   {
      char str[50];
      sprintf(str, "%d", _whichVessel + 1);
      XmTextFieldSetString(_magicDeck -> get_textfiedVessel(),  str);

      sprintf(str, "%7.3f", d);
      XmTextFieldSetString(_magicDeck -> get_textfiedVesselMeasure(),  str);
   }
}

void MagicBB::image_info(float d)
{
   _distance = d;
   if(d >= 0)
```

```
if( (fp=fopen(".fileOfFile", "r")) == NULL )
{
  return;
}

CanvasString *cs = new CanvasString();
List <CanvasString>  *files = new List <CanvasString>;

while(!feof(fp))
{
  fscanf(fp, "%s", cmd);
  cs -> set(cmd);
  cs -> trim(' ');
  cs -> trim('\n');
  cs -> trim('\t');
  if(files -> member(*cs) == 0)
    files -> insert(*cs);
}

ImgBase <GEAngio> *aux2 = new ImgBase <GEAngio>;
int  numMed = files -> length();
printf(" numMed %d\n", numMed);

GEAngio *mimg = aux2 -> alloc_1Ddata(numMed);


int x = 0;
CanvasString aFile;
if(files -> gotoBeginning())
{
  do
  {
    aFile = files -> getCursor();
    strcpy(cmd, dir -> get());
    strcat(cmd, aFile.get());
    if(mimg[x].open_file(flag, cmd))
    {
      printf("  %d   read_file succeed!  %s\n", x, cmd);
      mimg[x].init_grayImg();
      if(flag == 0)
        mimg[x].update_grayImg(0.0, 256.0);
      else if(flag == 1)
        mimg[x].update_grayImg(300.0, 1024.0);
      else if(flag == 2)
      {
        if(aFile.equal("cd.001"))
          mimg[x].update_grayImg(0, 0);
        else if(aFile.equal("mag.001"))
          mimg[x].update_grayImg(0, 0);
        else if(aFile.equal("pd.001"))
          mimg[x].update_grayImg(0, 0);
        else if(aFile.equal("perpcd.001"))
          mimg[x].update_grayImg(0, 0);
      }
      x++;
    }
  } while( files -> gotoNext() ); // && x <= 3);
}

/*
int x = 0;
CanvasString aFile;
if(files -> gotoBeginning())
{
  do
  {
```

```
    //---- Start editable code block: MagicBBUI resource table

      // { "resourceName"    setAttribute", XmRString},
    //---- End editable code block: MagicBBUI resource table
      { NULL }, // MUST be NULL terminated
    };

    return map;
} // End RegisterMagicBBInterface()




//---- End of generated code

//---- Start editable code block: End of generated code

void MagicBB::init()
{
    FILE *fp;

    if( (fp=fopen(".input", "r")) == NULL )
    {
      return;
    }
    char filename[300], company[100];
    fscanf(fp, "%s", filename);
    fscanf(fp, "%s", company);
    fclose(fp);
    CanvasString *companyCS = new CanvasString(company);

    int flag = 0;
    if(companyCS->equal("GE"))
      flag = 0;
    else if(companyCS->equal("SIEMENS"))
      flag = 1;
    else if(companyCS->equal("GEMRRAW"))
      flag = 2;
    else if(companyCS->equal("SCAN"))
      flag = 3;

    _blow = BlowZoom;

    _draw = new DrawingArea(0, 0, 810, 810, "glwidget", _imageBB);
    ((DrawingArea *)_draw) -> set(this);
    _draw -> show();

    _magicDeck -> set(this, _draw);

    int numH, numV;
    numH = numV = 5;
    ((DrawingArea *)_draw) -> set_layout(numH, numV);

    //CanvasString *dir = new CanvasString("/usr/people/meide/images/angio/");

    CanvasString *dir = new CanvasString(filename);

    char cmd[300];
    strcpy(cmd, "ls -FA ");
    strcat(cmd, dir -> get());
    strcat(cmd, " >.fileOfFile");
    system(cmd);
    usleep(500);
```

```
    //---- End editable code block: MagicBB setToggleZoom

}    // End MagicBB::setToggleZoom()



//////////////////////////////////////////////////////////////////////
// static creation function, for importing class into rapidapp
// or dynamically loading, using VkComponent::loadComponent
//////////////////////////////////////////////////////////////////////


VkComponent *MagicBB::CreateMagicBB( const char *name, Widget parent )
{
    VkComponent *obj =  new MagicBB ( name, parent );
    return ( obj );
} // End CreateMagicBB


//////////////////////////////////////////////////////////////////////
// Function for accessing a description of the dynamic interface
// to this class.
//////////////////////////////////////////////////////////////////////


// WARNING: This structure is different than that used with 1.1 RapidApp.
// See the RapidApp release notes for details

struct InterfaceMap {
  char  *resourceName;
  char  *methodName;
  char  *argType;
  char  *definingClass; // Optional, if not this class
  void (VkCallbackObject::*method)(...); // Reserved, do not set
};


void *MagicBB::RegisterMagicBBInterface()
{
    // This structure registers information about this class
    // that allows RapidApp to create and manipulate an instance.
    // Each entry provides a resource name that will appear in the
    // resource manager palette when an instance of this class is
    // selected, the name of the member function as a string,
    // the type of the single argument to this function, and an.
    // optional argument indicating the class that defines this function.
    // All member functions must have the form
    //
    //     void memberFunction ( Type );
    //
    // where "Type" is one of:
    //    const char *     (Use XmRString)
    //    Boolean          (Use XmRBoolean)
    //    int              (Use XmRInt)
    //    float            (Use XmRFloat)
    //    No argument      (Use VkRNoArg or "NoArg"
    //    A filename       (Use VkRFilename or "Filename")
    //    An enumeration   (Use "Enumeration:ClassName:Type: VALUE1, VALUE2, VALUE3")
    //    A callback       (Use XmRCallback)


    static InterfaceMap map[] = {
```

```
    //---- Start editable code block: MagicBB destructor

    //---- End editable code block: MagicBB destructor


}    // End Destructor

const char * MagicBB::className() // classname
{
    return ("MagicBB");
} // End className()

void MagicBB::setToggleAuto ( Widget w, XtPointer callData )
{
    //---- Start editable code block: MagicBB setToggleAuto

    XmToggleButtonCallbackStruct *cbs = (XmToggleButtonCallbackStruct*) callData;

    //--- Comment out the following line when MagicBB::setToggleAuto is implemented:

    //::VkUnimplemented ( w, "MagicBB::setToggleAuto" );

    if(_blow != BlowAuto) set_blow(BlowAuto);


    //---- End editable code block: MagicBB setToggleAuto

}    // End MagicBB::setToggleAuto()

void MagicBB::setToggleManual ( Widget w, XtPointer callData )
{
    //---- Start editable code block: MagicBB setToggleManual

    XmToggleButtonCallbackStruct *cbs = (XmToggleButtonCallbackStruct*) callData;

    //--- Comment out the following line when MagicBB::setToggleManual is implemented:

    //::VkUnimplemented ( w, "MagicBB::setToggleManual" );

    if(_blow != BlowDistance) set_blow(BlowDistance);


    //---- End editable code block: MagicBB setToggleManual

}    // End MagicBB::setToggleManual()

void MagicBB::setToggleZoom ( Widget w, XtPointer callData )
{
    //---- Start editable code block: MagicBB setToggleZoom

    XmToggleButtonCallbackStruct *cbs = (XmToggleButtonCallbackStruct*) callData;

    //--- Comment out the following line when MagicBB::setToggleZoom is implemented:

    //::VkUnimplemented ( w, "MagicBB::setToggleZoom" );

    if(_blow != BlowZoom) set_blow(BlowZoom);
```

```
//---- Start editable code block: headers and declarations

#include "DrawingArea.h"
#include "GEAngio.h"
#include "ImgBase.h"
#include "Listlnk.h"
#include "CanvasString.h"
#include <unistd.h>
#include "ModelDraw.h"
#include <Xm/TextF.h>
#include <math.h>
#include <stdlib.h>
#include "DispBB.h"

//---- End editable code block: headers and declarations


//---- MagicBB Constructor

MagicBB::MagicBB(const char *name, Widget parent) :
                    MagicBBUI(name, parent)
{
    // This constructor calls MagicBBUI(parent, name)
    // which calls MagicBBUI::create() to create
    // the widgets for this component. Any code added here
    // is called after the component's interface has been built

    //---- Start editable code block: MagicBB constructor

    init();

    //---- End editable code block: MagicBB constructor


}    // End Constructor


MagicBB::MagicBB(const char *name) :
                    MagicBBUI(name)
 {
    // This constructor calls MagicBBUI(name)
    // which does not create any widgets. Usually, this
    // constructor is not used

    //---- Start editable code block: MagicBB constructor 2

    init();

    //---- End editable code block: MagicBB constructor 2


}    // End Constructor


MagicBB::~MagicBB()
{
    // The base class destructors are responsible for
    // destroying all widgets and objects used in this component.
    // Only additional items created directly in this class
    // need to be freed here.
```

```
////////////////////////////////////////////////////////////
//
// Source file for MagicBB
//
//    This file is generated by RapidApp 1.2
//
//    This class is derived from MagicBBUI which
//    implements the user interface created in
//    RapidApp. This class contains virtual
//    functions that are called from the user interface.
//
//    When you modify this source, limit your changes to
//    modifying the sections between the
//    "//---- Start/End editable code block" markers
//
//    This will allow RapidApp to integrate changes more easily
//
//    This class is a ViewKit user interface "component".
//    For more information on how components are used, see the
//    "ViewKit Programmers' Manual", and the RapidApp
//    User's Guide.
////////////////////////////////////////////////////////////

#include "MagicBB.h"
#include <Vk/VkEZ.h>
#include <Xm/BulletinB.h>
#include <Xm/Label.h>
#include <Xm/RowColumn.h>
#include <Xm/ScrolledW.h>
#include <Xm/Separator.h>
#include <Xm/TextF.h>
#include <Xm/ToggleB.h>
#include <Vk/VkResource.h>


// Externally defined classes referenced by this class:

#include "MagicDeckTabbedDeck.h"
#include <Vk/VkSimpleWindow.h>


extern void VkUnimplemented ( Widget, const char * );

////////////////////////////////////////////////////////////////////////////
// The following non-container elements are created by MagicBBUI and are
// available as protected data members inherited by this class
//
//  XmScrolledWindow                _scrolledModel
//  XmTextField             _textfieldAnatomy
//  XmToggleButton                  _toggleZoom
//  XmToggleButton                  _toggleAutomatic
//  XmToggleButton                  _toggleManual
//  XmLabel                 _labelPixels
//  XmLabel                 _labelDistance
//  XmTextField             _textfieldDistance
//  XmTextField             _textfieldPixels
//  XmSeparator             _magicSep
//
// The following components are created by MagicBBUI and are
// available as protected data members inherited by this class
//
//  MagicDeckTabbedDeck             *_magicDeck
//
////////////////////////////////////////////////////////////////////////////
```

```
protected:

    // These functions will be called as a result of callbacks
    // registered in MagicBBUI

    virtual void setToggleAuto ( Widget, XtPointer );
    virtual void setToggleManual ( Widget, XtPointer );
    virtual void setToggleZoom ( Widget, XtPointer );

    //---- Start editable code block: MagicBB protected

    //---- End editable code block: MagicBB protected


private:

    static void* RegisterMagicBBInterface();

    //---- Start editable code block: MagicBB private


    //---- End editable code block: MagicBB private


};
//---- Start editable code block: End of generated code


//---- End editable code block: End of generated code

#endif
```

```
////////////////////////////////////////////////////////////
//
// Header file for MagicBB
//
//    This file is generated by RapidApp 1.2
//
//    This class is derived from MagicBBUI which
//    implements the user interface created in
//    RapidApp. This class contains virtual
//    functions that are called from the user interface.
//
//    When you modify this header file, limit your changes to those
//    areas between the "//---- Start/End editable code block" markers
//
//    This will allow RapidApp to integrate changes more easily
//
//    This class is a ViewKit user interface "component".
//    For more information on how components are used, see the
//    "ViewKit Programmers' Manual", and the RapidApp
//    User's Guide.
////////////////////////////////////////////////////////////
#ifndef MAGICBB_H
#define MAGICBB_H
#include "MagicBBUI.h"
//---- Start editable code block: headers and declarations


//---- End editable code block: headers and declarations


//---- MagicBB class declaration

class MagicBB : public MagicBBUI
{

  public:

    MagicBB ( const char *, Widget );
    MagicBB ( const char * );
    ~MagicBB();
    const char *  className();

    static VkComponent *CreateMagicBB( const char *name, Widget parent );

    //---- Start editable code block: MagicBB public

    int    _blow;
    void set_blow(int);

    void   init();
    class DrawingArea  *_draw;
    void vessel_info(int, float);

    float  _distance;
    void image_info(float);

    int  _whichVessel;
    void set_measured(float d);

    void anatomy_info(char *a);

    class ModelDraw   *_modelDrawArea;

    //---- End editable code block: MagicBB public
```

```
    MagicBBUI* obj = ( MagicBBUI * ) clientData;
    obj->setToggleZoom ( [redacted]callData );
}


//////////////////////////////////////////////////////////////
// The following functions are called from the menu items
// in this window.
//////////////////////////////////////

void MagicBBUI::setToggleAuto ( Widget, XtPointer )
{
    // This virtual function is called from setToggleAutoCallback.
    // This function is normally overriden by a derived class.

}

void MagicBBUI::setToggleManual ( Widget, XtPointer )
{
    // This virtual function is called from setToggleManualCallback.
    // This function is normally overriden by a derived class.

}

void MagicBBUI::setToggleZoom ( Widget, XtPointer )
{
    // This virtual function is called from setToggleZoomCallback.
    // This function is normally overriden by a derived class.

}


//---- Start editable code block: End of generated code


//---- End editable code block: End of generated code
```

```
                                        XmNx, 40,
                                        XmNy,
                                        XmNwid    90,
                                        XmNheight, 40,
                                        (XtPointer) NULL );

    _magicDeck = new MagicDeckTabbedDeck( "magicDeck", _baseWidget  );
    _magicDeck->show();

    _magicSep = XtVaCreateManagedWidget  ( "magicSep",
                                           xmSeparatorWidgetClass,
                                           _baseWidget,
                                           XmNorientation, XmVERTICAL,
                                           XmNx, 424,
                                           XmNy, 10,
                                           XmNwidth, 20,
                                           XmNheight, 960,
                                           (XtPointer) NULL );


    XtVaSetValues ( _magicDeck->baseWidget(),
                    XmNx, 10,
                    XmNy, 554,
                    XmNwidth, 420,
                    XmNheight, 420,
                    (XtPointer) NULL );

    //---- Start editable code block: MagicBBUI create

    //---- End editable code block: MagicBBUI create
}

const char * MagicBBUI::className()
{
    return ("MagicBBUI");
}    // End className()



////////////////////////////////////////////////////////////////
// The following functions are static member functions used to
// interface with Motif.
////////////////////////////////////

void MagicBBUI::setToggleAutoCallback ( Widget    w,
                                        XtPointer clientData,
                                        XtPointer callData )
{
    MagicBBUI* obj = ( MagicBBUI * ) clientData;
    obj->setToggleAuto ( w, callData );
}

void MagicBBUI::setToggleManualCallback ( Widget    w,
                                          XtPointer clientData,
                                          XtPointer callData )
{
    MagicBBUI* obj = ( MagicBBUI * ) clientData;
    obj->setToggleManual ( w, callData );
}

void MagicBBUI::setToggleZoomCallback ( Widget    w,
                                        XtPointer clientData,
                                        XtPointer callData )
{
```

```
XtAddCallback ( _togg[redacted]oom,
                XmNva[redacted]eChangedCallback,
                &MagicBBUI::setToggleZoomCallback,
                (XtPointer) this );


_toggleAutomatic = XtVaCreateManagedWidget  ( "toggleAutomatic",
                                              xmToggleButtonWidgetClass,
                                              _radiobox,
                                              XmNlabelType, XmSTRING,
                                              (XtPointer) NULL );

XtAddCallback ( _toggleAutomatic,
                XmNvalueChangedCallback,
                &MagicBBUI::setToggleAutoCallback,
                (XtPointer) this );


_toggleManual = XtVaCreateManagedWidget  ( "toggleManual",
                                           xmToggleButtonWidgetClass,
                                           _radiobox,
                                           XmNlabelType, XmSTRING,
                                           (XtPointer) NULL );

XtAddCallback ( _toggleManual,
                XmNvalueChangedCallback,
                &MagicBBUI::setToggleManualCallback,
                (XtPointer) this );


_labelPixels = XtVaCreateManagedWidget  ( "labelPixels",
                                          xmLabelWidgetClass,
                                          _imageBB,
                                          XmNlabelType, XmSTRING,
                                          XmNx, 149,
                                          XmNy, 927,
                                          XmNwidth, 36,
                                          XmNheight, 20,
                                          (XtPointer) NULL );


_labelDistance = XtVaCreateManagedWidget  ( "labelDistance",
                                            xmLabelWidgetClass,
                                            _imageBB,
                                            XmNlabelType, XmSTRING,
                                            XmNx, 150,
                                            XmNy, 879,
                                            XmNwidth, 28,
                                            XmNheight, 20,
                                            (XtPointer) NULL );


_textfieldDistance = XtVaCreateManagedWidget  ( "textfieldDistance",
                                                xmTextFieldWidgetClass,
                                                _imageBB,
                                                XmNx, 40,
                                                XmNy, 869,
                                                XmNwidth, 90,
                                                XmNheight, 40,
                                                (XtPointer) NULL );


_textfieldPixels = XtVaCreateManagedWidget  ( "textfieldPixels",
                                              xmTextFieldWidgetClass,
                                              _imageBB,
```

```
_baseWidget = _magicBB = XtVaCreateWidget ( _name,
                                            xmBulleti[redacted]ardWidgetClass,
                                            parent,
                                            XmNresizePolicy, XmRESIZE_GROW,
                                            (XtPointer) NULL );

// install a callback to guard against unexpected widget destruction

installDestroyHandler();


// Create widgets used in this component
// All variables are data members of this class

_scrolledModel = XtVaCreateManagedWidget  ( "scrolledModel",
                                             xmScrolledWindowWidgetClass,
                                             _baseWidget,
                                             XmNscrollingPolicy, XmAUTOMATIC,
                                             XmNscrollBarDisplayPolicy, XmAS_NEEDEI
                                             XmNx, 10,
                                             XmNy, 10,
                                             XmNwidth, 414,
                                             XmNheight, 534,
                                             (XtPointer) NULL );


_imageBB = XtVaCreateManagedWidget  ( "imageBB",
                                      xmBulletinBoardWidgetClass,
                                      _baseWidget,
                                      XmNresizePolicy, XmRESIZE_GROW,
                                      XmNx, 444,
                                      XmNy, 10,
                                      XmNwidth, 810,
                                      XmNheight, 970,
                                      (XtPointer) NULL );


_textfieldAnatomy = XtVaCreateManagedWidget  ( "textfieldAnatomy",
                                               xmTextFieldWidgetClass,
                                               _imageBB,
                                               XmNx, 550,
                                               XmNy, 900,
                                               XmNwidth, 250,
                                               XmNheight, 40,
                                               (XtPointer) NULL );


_radiobox = XtVaCreateManagedWidget  ( "radiobox",
                                       xmRowColumnWidgetClass,
                                       _imageBB,
                                       XmNorientation, XmHORIZONTAL,
                                       XmNpacking, XmPACK_COLUMN,
                                       XmNradioBehavior, True,
                                       XmNradioAlwaysOne, True,
                                       XmNx, 255,
                                       XmNy, 900,
                                       XmNwidth, 231,
                                       XmNheight, 32,
                                       (XtPointer) NULL );


_toggleZoom = XtVaCreateManagedWidget  ( "toggleZoom",
                                         xmToggleButtonWidgetClass,
                                         _radiobox,
                                         XmNlabelType, XmSTRING,
                                         (XtPointer) NULL );
```

```
{
    // No widgets are created by this constructor.
    // If an application creates a component using this constructor,
    // It must explictly call create at a later time.
    // This is mostly useful when adding pre-widget creation
    // code to a derived class constructor.

    //---- Start editable code block: MagicBB constructor 2


    //---- End editable code block: MagicBB constructor 2


}    // End Constructor



MagicBBUI::MagicBBUI ( const char *name, Widget parent ) : VkComponent ( name )
{
    //---- Start editable code block: MagicBB pre-create


    //---- End editable code block: MagicBB pre-create


    // Call creation function to build the widget tree.

     create ( parent );

    //---- Start editable code block: MagicBB constructor


    //---- End editable code block: MagicBB constructor


}    // End Constructor

MagicBBUI::~MagicBBUI()
{
    delete _magicDeck;


    //---- Start editable code block: MagicBBUI destructor


    //---- End editable code block: MagicBBUI destructor
}    // End destructor


void MagicBBUI::create ( Widget parent )
{
    Arg      args[8];
    Cardinal count;
    count = 0;

    // Load any class-defaulted resources for this object

    setDefaultResources ( parent, _defaultMagicBBUIResources  );


    // Create an unmanaged widget as the top of the widget hierarchy
```

```
////////////////////////////////////////////////////////////
//
// Source file for MagicBBUI
//
//    This class implements the user interface created in
//    RapidApp.
//
//    Restrict changes to those sections between
//    the "//--- Start/End editable code block" markers
//
//    This will allow RapidApp to integrate changes more easily
//
//    This class is a ViewKit user interface "component".
//    For more information on how components are used, see the
//    "ViewKit Programmers' Manual", and the RapidApp
//    User's Guide.
//
//
////////////////////////////////////////////////////////////


#include "MagicBBUI.h" // Generated header file for this class

#include <Xm/BulletinB.h>
#include <Xm/Label.h>
#include <Xm/RowColumn.h>
#include <Xm/ScrolledW.h>
#include <Xm/Separator.h>
#include <Xm/TextF.h>
#include <Xm/ToggleB.h>
#include <Vk/VkResource.h>


// Externally defined classes referenced by this class:

#include "MagicDeckTabbedDeck.h"
//---- Start editable code block: headers and declarations

#include <stdio.h>

//---- End editable code block: headers and declarations


// These are default resources for widgets in objects of this class
// All resources will be prepended by *<name> at instantiation,
// where <name> is the name of the specific instance, as well as the
// name of the baseWidget. These are only defaults, and may be overriden
// in a resource file by providing a more specific resource name

String  MagicBBUI::_defaultMagicBBUIResources[] = {
        "*labelDistance.labelString:  mm",
        "*labelPixels.labelString:  pixel",
        "*toggleAutomatic.labelString:  Auto",
        "*toggleManual.labelString:  Manual",
        "*toggleZoom.labelString:  Zoom",

        //---- Start editable code block: MagicBBUI Default Resources


        //---- End editable code block: MagicBBUI Default Resources

        (char*)NULL
};

MagicBBUI::MagicBBUI ( const char *name ) : VkComponent ( name )
```

```
    Widget  _imageBB;
    Widget  _labelDistanc
    Widget  _labelPixels;
    Widget  _magicBB;
    Widget  _magicSep;
    Widget  _radiobox;
    Widget  _scrolledModel;
    Widget  _textfieldAnatomy;
    Widget  _textfieldDistance;
    Widget  _textfieldPixels;
    Widget  _toggleAutomatic;
    Widget  _toggleManual;
    Widget  _toggleZoom;


    // These virtual functions are called from the private callbacks (below)
    // Intended to be overriden in derived classes to define actions

    virtual void setToggleAuto ( Widget, XtPointer );
    virtual void setToggleManual ( Widget, XtPointer );
    virtual void setToggleZoom ( Widget, XtPointer );

    //---- Start editable code block: MagicBB protected

    //---- End editable code block: MagicBB protected


  private:

    // Array of default resources

    static String      _defaultMagicBBUIResources[];


    // Callbacks to interface with Motif

    static void setToggleAutoCallback ( Widget, XtPointer, XtPointer );
    static void setToggleManualCallback ( Widget, XtPointer, XtPointer );
    static void setToggleZoomCallback ( Widget, XtPointer, XtPointer );

    //---- Start editable code block: MagicBB private

    //---- End editable code block: MagicBB private
};
//---- Start editable code block: End of generated code


//---- End editable code block: End of generated code

#endif
```

```
////////////////////////////////////////////////////////////
//
// Header file for MagicBBUI
//
//    This file is generated by RapidApp 1.2
//
//    This class implements the user interface portion of a class
//    Normally it is not used directly.
//    Instead the subclass, MagicBB is instantiated
//
//    To extend or alter the behavior of this class, you should
//    modify the MagicBB files
//
//    Restrict changes to those sections between
//    the "//--- Start/End editable code block" markers
//
//    This will allow RapidApp to integrate changes more easily
//
//    This class is a ViewKit user interface "component".
//    For more information on how components are used, see the
//    "ViewKit Programmers' Manual", and the RapidApp
//    User's Guide.
//
////////////////////////////////////////////////////////////
#ifndef MAGICBBUI_H
#define MAGICBBUI_H
#include <Vk/VkComponent.h>


//---- Start editable code block: headers and declarations


//---- End editable code block: headers and declarations


// Externally defined classes referenced by this class:

class MagicDeckTabbedDeck;


class MagicBBUI : public VkComponent
{

  public:

    MagicBBUI ( const char *, Widget );
    MagicBBUI ( const char * );
    ~MagicBBUI();
    void create ( Widget );
    const char *  className();

    //---- Start editable code block: MagicBB public

    //---- End editable code block: MagicBB public


  protected:


    // Classes created by this class

    class MagicDeckTabbedDeck *_magicDeck;

    // Widgets created by this class
```

```
    setDefaultResources ( [redacted]seWidget(), _defaultMagicDeck[redacted]bedDeckResources );

    _measureBB = new MeasureBB( "measureBB", _vkdeck->baseWidget()  );
    _measureBB->show();

    _dispBB = new DispBB( "dispBB", _vkdeck->baseWidget()  );
    _dispBB->show();

    _calibBB = new CalibBB( "calibBB", _vkdeck->baseWidget()  );
    _calibBB->show();


    registerChild ( _measureBB, "measureBB");
    registerChild ( _dispBB, "dispBB");
    registerChild ( _calibBB, "calibBB");

    //---- Start editable code block: MagicDeckTabbedDeck constructor

    //---- End editable code block: MagicDeckTabbedDeck constructor


}


MagicDeckTabbedDeck::~MagicDeckTabbedDeck()
{
    //---- Start editable code block: MagicDeckTabbedDeck destructor


    //---- End editable code block: MagicDeckTabbedDeck destructor

}


const char * MagicDeckTabbedDeck::className() // classname
{
    return ("MagicDeckTabbedDeck");
} // End className()




//---- Start editable code block: End of generated code

void MagicDeckTabbedDeck::set(class VkComponent *v, class VkComponent *v1)
{
    _parent = v;
    _dispBB -> set(v1);
    _calibBB -> set(this);
    _measureBB -> set(v);
}

//---- End editable code block: End of generated code
```

```
//////////////////////////////////////////////////////////////
//
// Source file for MagicDeckTabbedDeck
//
//    This file is generated by RapidApp 1.2
//
//    This class is derived from VkTabbedDeck
//    When you modify this source, limit your changes to
//    modifying the sections between the
//    "//---- Start/End editable code block" markers
//
//    This will allow the builder to integrate changes more easily
//
//    This class is a ViewKit user interface "component".
//    For more information on how components are used, see the
//    "ViewKit Programmers' Manual", and the RapidApp
//    User's Guide.
//////////////////////////////////////////////////////////////

#include "MagicDeckTabbedDeck.h"
#include <Vk/VkDeck.h>

#include <Vk/VkResource.h>


// Externally defined classes referenced by this class:

#include "CalibBB.h"
#include "DispBB.h"
#include "MeasureBB.h"
extern void VkUnimplemented(Widget, const char *);


//---- Start editable code block: headers and declarations

#include <Xm/TextF.h>
#include <stdio.h>
#include "MagicBB.h"

//---- End editable code block: headers and declarations


// These are default resources for widgets in objects of this class
// All resources will be prepended by *<name> at instantiation,
// where <name> is the name of the specific instance, as well as the
// name of the baseWidget. These are only defaults, and may be overriden
// in a resource file by providing a more specific resource name

String  MagicDeckTabbedDeck::_defaultMagicDeckTabbedDeckResources[] = {

        //---- Start editable code block: MagicDeckTabbedDeck Default Resources


        //---- End editable code block: MagicDeckTabbedDeck Default Resources

        (char*)NULL
};


//---- MagicDeckTabbedDeck Constructor

MagicDeckTabbedDeck::MagicDeckTabbedDeck ( const char *name, Widget parent ) :
                VkTabbedDeck ( name, parent )
{
    // Load any class-default resources for this object
```

```
    // Classes created by this class

    class MeasureBB *_measureBB;
    class DispBB *_dispBB;
    class CalibBB *_calibBB;

    // Widgets created by this class

    Widget  _magicDeck;

    //---- Start editable code block: MagicDeckTabbedDeck protected


    //---- End editable code block: MagicDeckTabbedDeck protected


  private:

    // Array of default resources

    static String      _defaultMagicDeckTabbedDeckResources[];

    //---- Start editable code block: MagicDeckTabbedDeck private


    //---- End editable code block: MagicDeckTabbedDeck private
};
//---- Start editable code block: End of generated code


//---- End editable code block: End of generated code

#endif
```

```
//////////////////////////////////////////////////////////////
//
// Header file for MagicDeckTabbedDeck
//
//    This file is generated by RapidApp 1.2
//
//    This class is derived from VkTabbedDeck
//
//    When you modify this header file, limit your changes to those
//    areas between the "//---- Start/End editable code block" markers
//
//    This will allow the builder to integrate changes more easily
//
//    This class is a ViewKit user interface "component".
//    For more information on how components are used, see the
//    "ViewKit Programmers' Manual", and the RapidApp
//    User's Guide.
//////////////////////////////////////////////////////////////
#ifndef MAGICDECKTABBEDDECK_H
#define MAGICDECKTABBEDDECK_H
#include <Vk/VkTabbedDeck.h>
//---- Start editable code block: headers and declarations

#include "MeasureBBUI.h"
#include "CalibBBUI.h"


//---- End editable code block: headers and declarations



//---- MagicDeckTabbedDeck class declaration

class MagicDeckTabbedDeck : public VkTabbedDeck
{

  public:

    MagicDeckTabbedDeck ( const char *, Widget );
    MagicDeckTabbedDeck ( const char * );
    ~MagicDeckTabbedDeck();
    const char *  className();

    static VkComponent *CreateMagicDeckTabbedDeck( const char *name, Widget parent );

    //---- Start editable code block: MagicDeckTabbedDeck public

    class VkComponent *_parent;
    void set(class VkComponent *v, class VkComponent *v1);

    Widget  get_textfiedVessel() {return ((MeasureBBUI *)_measureBB) -> _textfiedVessel
    Widget  get_textfiedVesselMeasure() {return ((MeasureBBUI *)_measureBB) -> _textfie

    Widget  get_textfieldImagePixel() {return ((MeasureBBUI *)_measureBB) -> _textfield
    Widget  get_textfieldImageDistance() {return ((MeasureBBUI *)_measureBB) -> _textfi

    Widget  get_textfieldPixsize() {return ((CalibBBUI *)_calibBB) -> _textfieldPixsize

    //---- End editable code block: MagicDeckTabbedDeck public


    protected:
```

```
#include "ImgBase.h"

MagicGrid::MagicGrid()
{
  _img = NULL;
}

MagicGrid <LE> ::~MagicGrid()
{

}

void MagicGrid::init(int i, int j, int numH, int numV,  int w,  int h, int gapH, int ga
{
   if(w != 0 && h != 0)
   {
    _widthB = int(float(w - 1)/numH);
    _heightB = int(float(h - 1)/numV);
    _widthI =  _widthB - gapH;
    _heighI =  _heightB - gapV;

    _xboard1 = _widthB * i;

    _yboard1 =   _heightB * numV -  _heightB * (j + 1);
    _xboard2 =  _xboard1 +  _widthB;
    _yboard2 =  _yboard1 +  _heightB;

    _ximg1 =  _xboard1 + gapH/2.0;
    _yimg1 =  _yboard1 + gapV/2.0;
    _ximg2 =  _ximg1 +  _widthI;
    _yimg2 =  _yimg1 +  _heighI;
  }
  else
  {
    _widthB = _heightB = _widthI = _heighI = 0;
    _xboard1 = _yboard1 = _xboard2 = _yboard2 = 0;
    _ximg1 = _yimg1 = _ximg2 = _ximg1 = _yimg2 = 0;
  }
}

int MagicGrid::isWithin(int i, int j, int xpos,  int ypos)
{
    x1 =   _widthB * i;
    y1 =   _heightB * j;
    x2 = x1 +  _widthB;
    y2 = y1 +  _heightB;
    if(xpos >= x1 && xpos <= x2 && ypos >= y1 && ypos <= y2)
      return 1;
    else
      return 0;
}
```

```
////////////////////////////////////////////////////////////
// The following functions are called from callbacks
////////////////////////////////////////////////////////////


//---- Start editable code block: End of generated code


//---- End editable code block: End of generated code
```

```
{
    // Load any class-default resources for this object
    setDefaultResources ( baseWidget(), _defaultMagicWinMainWindowResources );

    // Create the view component contained by this window

    _magicBB = new MagicBB ( "magicBB",mainWindowWidget() );

    XtVaSetValues ( _magicBB->baseWidget(),
                    XmNwidth, 1264,
                    XmNheight, 984,
                    (XtPointer) NULL );

    // Add the component as the main view

    addView ( _magicBB );

    //---- Start editable code block: MagicWinMainWindow constructor


    //---- End editable code block: MagicWinMainWindow constructor


}    // End Constructor


MagicWinMainWindow::~MagicWinMainWindow()
{
    delete _magicBB;
    //---- Start editable code block: MagicWinMainWindow destructor


    //---- End editable code block: MagicWinMainWindow destructor
}    // End destructor


const char *MagicWinMainWindow::className()
{
    return ("MagicWinMainWindow");
}    // End className()


Boolean MagicWinMainWindow::okToQuit()
{
    //---- Start editable code block: MagicWinMainWindow okToQuit



    // This member function is called when the user quits by calling
    // theApplication->terminate() or uses the window manager close protocol
    // This function can abort the operation by returning FALSE, or do some.
    // cleanup before returning TRUE. The actual decision is normally passed on
    // to the view object

    return TRUE;
    // Query the view object, and give it a chance to cleanup

    //---- End editable code block: MagicWinMainWindow okToQuit
}    // End okToQuit()
```

```
//////////////////////////////////////////////////////////////////////
//
// Source file for MagicWinMainWindow
//
//    This class is a subclass of VkSimpleWindow
//
//
// Normally, very little in this file should need to be changed.
// Create/add/modify menus using RapidApp.
//
// Try to restrict any changes to the bodies of functions
// corresponding to menu items, the constructor and destructor.
//
// Restrict changes to those sections between
// the "//--- Start/End editable code block" markers
//
// Doing so will allow you to make changes using RapidApp
// without losing any changes you may have made manually
//
// Avoid gratuitous reformatting and other changes that might
// make it difficult to integrate changes made using RapidApp
//////////////////////////////////////////////////////////////////////
#include "MagicWinMainWindow.h"

#include <Vk/VkApp.h>
#include <Vk/VkResource.h>


// Externally defined classes referenced by this class:

#include "MagicBB.h"

extern void VkUnimplemented ( Widget, const char * );

//---- Start editable code block: headers and declarations


//---- End editable code block: headers and declarations



// These are default resources for widgets in objects of this class
// All resources will be prepended by *<name> at instantiation,
// where <name> is the name of the specific instance, as well as the
// name of the baseWidget. These are only defaults, and may be overriden
// in a resource file by providing a more specific resource name

String  MagicWinMainWindow::_defaultMagicWinMainWindowResources[] = {
        "*title:  CANVAS",

        //---- Start editable code block: MagicWinMainWindow Default Resources


        //---- End editable code block: MagicWinMainWindow Default Resources

        (char*)NULL
};


//---- Class declaration

MagicWinMainWindow::MagicWinMainWindow ( const char *name,
                                         ArgList args,
                                         Cardinal argCount) :
                                    VkSimpleWindow ( name, args, argCount )
```

```
    static String  _defau[illegible]agicWinMainWindowResources[];

    //---- Start editable code block: MagicWinMainWindow private

    //---- End editable code block: MagicWinMainWindow private

};
//---- Start editable code block: End of generated code

//---- End editable code block: End of generated code

#endif
```

```
////////////////////////////////////////////////////////////////
//
// Header file for MagicWinMainWindow
//
//    This class is a subclass of VkSimpleWindow
//
// Normally, very little in this file should need to be changed.
// Create/add/modify menus using RapidApp.
//
// Restrict changes to those sections between
// the "//--- Start/End editable code block" markers
// Doing so will allow you to make changes using RapidApp
// without losing any changes you may have made manually
//
////////////////////////////////////////////////////////////////
#ifndef MAGICWINMAINWINDOW_H
#define MAGICWINMAINWINDOW_H
#include <Vk/VkSimpleWindow.h>


//---- Start editable code block: headers and declarations


//---- End editable code block: headers and declarations


//---- MagicWinMainWindow class declaration

class MagicWinMainWindow: public VkSimpleWindow {

  public:

    MagicWinMainWindow( const char * name,
                        ArgList args = NULL,
                        Cardinal argCount = 0 );
    ~MagicWinMainWindow();
    const char *className();
    virtual Boolean okToQuit();

    //---- Start editable code block: MagicWinMainWindow public


    //---- End editable code block: MagicWinMainWindow public


  protected:


    // Classes created by this class

    class MagicBB *_magicBB;


    // Widgets created by this class


    //---- Start editable code block: MagicWinMainWindow protected


    //---- End editable code block: MagicWinMainWindow protected


  private:
```

```
TARGETS=magic
APPDEFAULTS=Magic
default all: $(TARGETS)


$(TARGETS): $(OBJECTS)
        $(C++F) $(OPTIMIZER) $(OBJECTS) $(LDFLAGS) -o $@
#
# These flags instruct the compiler to output
# analysis information for cvstatic
# Uncoment to enable
# Be sure to also disable smake if cvstatic is used

#SADIR= magicview.cvdb
#SAFLAG= -sa,$(SADIR)
#$(OBJECTS):$(SADIR)/cvdb.dbd
#$(SADIR)/cvdb.dbd :
#       [ -d $(SADIR) ] || mkdir $(SADIR)
#       cd $(SADIR); initcvdb.sh

#LDIRT=$(SADIR) vista.taf

print:  $(BUILDERFILES)
        lp -dLaserJet $(BUILDERFILES)

printh: $(HEADFILES)
        lp -dLaserJet $(HEADFILES)

#
# To install on the local machine, do 'make install'
#

install: all
        $(INSTALL) -F /usr/lib/X11/app-defaults Magic
        $(INSTALL) -F /usr/sbin magic
        $(INSTALL) -F /usr/lib/images Magic.icon

#
# To create inst images, do 'make image'
# An image subdirectory should already exist
#

image images: $(TARGETS)
        [ -d $(IMAGEDIR) ] || mkdir $(IMAGEDIR)
        /usr/sbin/gendist -rbase / -sbase  `pwd` -idb magic.idb \
         -spec magic.spec \
        -dist $(IMAGEDIR)  -all

include $(COMMONRULES)
```

```
##---- End editable code block: definitions

#  The GL library being used, if needed

GLLIBS=-lGLw -lGL -lGLU
COMPONENTLIBS=


#
# The ViewKit stub help library (-lvkhelp) provides a simple
# implementation of the SGI help API. Changing this to -lhelpmsg
# switches to the full IRIS Insight help system
#

HELPLIB= -lvkhelp

MESSAGELIBS=
LICENSELIB=

EZLIB = -lvkEZ
VIEWKITLIBS= $(MESSAGELIBS) $(EZLIB) -lvk $(HELPLIB) $(LICENSELIB) -lSgm -lXpm

# Local C++ options.
# woff 3262 shuts off warnings about arguments that are declared
# but not referenced.

WOFF= -woff 3262

LCXXOPTS = -nostdinc  -I. -I$(ROOT)/usr/include/CC -I$(ROOT)/usr/include $(SAFLAG) $(WC

LLDLIBS =  -L$(ROOT)/usr/lib $(USERLIBS) $(COMPONENTLIBS) $(VIEWKITLIBS) $(GLLIBS) -lXm


# SGI makefiles don't recognize all C++ sufixes, so set up
# the one being used here.

CXXO3=$(CXXO2:.C=.o)
CXXOALL=$(CXXO3)

#
# Source Files generated by RapidApp. If files are added
# manually, add them to USERFILES
#

BUILDERFILES =  main.C\
                CalibBB.C\
                CalibBBUI.C\
                DispBB.C\
                DispBBUI.C\
                MagicBB.C\
                MagicBBUI.C\
                MagicDeckTabbedDeck.C\
                MagicWinMainWindow.C\
                MeasureBB.C\
                MeasureBBUI.C\
                unimplemented.C\
                $(NULL)

C++FILES = $(BUILDERFILES) $(USERFILES)


#
# The program being built
#
```

```
#!smake
#
# Makefile for magic
# Generated by RapidApp 1.2
#
# This makefile follows various conventions used by SGI makefiles
# See the RapidApp User's Guide for more information
# This makefile supports most common default rules, including:
#   make (or make all):  build the application or library
#   make install:        install the application or library on the local machine
#   make image(s):       create "inst" images for distribution
#   make clean:          remove .o's, core, etc.
#   make clobber:        make clean + remove the target application.
# You should be able to customize this Makefile by editing
# only the section between the ##---- markers below.
# Specify additional files, compiler flags, libraries
# by changing the appropriate variables
include $(ROOT)/usr/include/make/commondefs

##---- Start editable code block: definitions

############################################################
############################################################
# Modify the following variables to customize this makefile
############################################################
############################################################
#
# Local Definitions
#

# Directory in which inst images are placed
# NOTE: do not name this directory 'image', as it will
# cause a cycle in the Makefile graph

IMAGEDIR= images

# Add Additional libraries to USERLIBS:

BASE=       /usr/people/meide/.susong/cmis100
BASELIB=    $(BASE)/lib

USERLIBS= -L$(BASELIB)/libGUI -lGUI -L$(BASELIB)/ydai -lYdai -lm

# While developing, leave OPTIMIZER set to -g.
# For delivery, change to -O2

OPTIMIZER= -g
#
# Add any files added outside RapidApp here
#

USERFILES = $(BASELIB)/libImages/GEAngio.C \
            $(BASELIB)/libImages/MedImage.C \
            $(BASELIB)/libGeneral/CanvasString.C \
            $(BASELIB)/libGeneral/Listlnk.C

HEADFILES = CalibBB.h CalibBBUI.h  DispBB.h DispBBUI.h MagicBB.h MagicBBUI.h \
            MagicDeckTabbedDeck.h MagicWinMainWindow.h MeasureBB.h \
            MeasureBBUI.h

#
# Add compiler flags here
#

USERFLAGS = -I$(BASELIB)/libGUI -I$(BASELIB)/libGeneral -I$(BASELIB)/libImages \
            -I$(BASELIB)/ydai
```

```
    //    float             (Use XmRFloat)
    //    No argument       (Use VkRNoArg or "NoArg"
    //    A filename        (Use VkRFilename or "Filename")
    //    An enumeration    (Use "Enumeration:ClassName:Type: VALUE1, VALUE2, VALUE3")
    //    A callback        (Use XmRCallback)


    static InterfaceMap map[] = {
    //---- Start editable code block: MeasureBBUI resource table

      // { "resourceName", "setAttribute", XmRString},
    //---- End editable code block: MeasureBBUI resource table
      { NULL }, // MUST be NULL terminated
    };

    return map;
} // End RegisterMeasureBBInterface()




//---- End of generated code

//---- Start editable code block: End of generated code

//---- End editable code block: End of generated code
```

```
    //::VkUnimplemented ( "MeasureBB::acceptMeasured" )

    float d = atof(XmTextFieldGetString(_textfieldImageDistance));

    ((MagicBB *)_parent) -> set_measured(d);
    char str[50];
    sprintf(str, "%8.3f", d);
    XmTextFieldSetString(_textfieldVesselMeasure, str);

    //---- End editable code block: MeasureBB acceptMeasured

}    // End MeasureBB::acceptMeasured()



////////////////////////////////////////////////////////////////////
// static creation function, for importing class into rapidapp
// or dynamically loading, using VkComponent::loadComponent
////////////////////////////////////////////////////////////////////

VkComponent *MeasureBB::CreateMeasureBB( const char *name, Widget parent )
{
    VkComponent *obj =  new MeasureBB ( name, parent );
    return ( obj );
} // End CreateMeasureBB


////////////////////////////////////////////////////////////////////
// Function for accessing a description of the dynamic interface
// to this class.
////////////////////////////////////////////////////////////////////

// WARNING: This structure is different than that used with 1.1 RapidApp.
// See the RapidApp release notes for details

struct InterfaceMap {
  char  *resourceName;
  char  *methodName;
  char  *argType;
  char  *definingClass; // Optional, if not this class
  void (VkCallbackObject::*method)(...); // Reserved, do not set
};


void *MeasureBB::RegisterMeasureBBInterface()
{
    // This structure registers information about this class
    // that allows RapidApp to create and manipulate an instance.
    // Each entry provides a resource name that will appear in the
    // resource manager palette when an instance of this class is
    // selected, the name of the member function as a string,
    // the type of the single argument to this function, and an.
    // optional argument indicating the class that defines this function.
    // All member functions must have the form
    //
    //     void memberFunction ( Type );
    //
    // where "Type" is one of:
    //    const char *     (Use XmRString)
    //    Boolean          (Use XmRBoolean)
    //    int              (Use XmRInt)
```

```
MeasureBB::MeasureBB(cons[redacted]ar *name, Widget parent) :
                    MeasureBBUI(name, parent)
{
    // This constructor calls MeasureBBUI(parent, name)
    // which calls MeasureBBUI::create() to create
    // the widgets for this component. Any code added here
    // is called after the component's interface has been built

    //---- Start editable code block: MeasureBB constructor

    //---- End editable code block: MeasureBB constructor


}    // End Constructor


MeasureBB::MeasureBB(const char *name) :
                    MeasureBBUI(name)
 {
    // This constructor calls MeasureBBUI(name)
    // which does not create any widgets. Usually, this
    // constructor is not used

    //---- Start editable code block: MeasureBB constructor 2

    //---- End editable code block: MeasureBB constructor 2


}    // End Constructor


MeasureBB::~MeasureBB()
{
    // The base class destructors are responsible for
    // destroying all widgets and objects used in this component.
    // Only additional items created directly in this class
    // need to be freed here.

    //---- Start editable code block: MeasureBB destructor


    //---- End editable code block: MeasureBB destructor



}    // End Destructor


const char * MeasureBB::className() // classname
{
    return ("MeasureBB");
} // End className()


void MeasureBB::acceptMeasured ( Widget w, XtPointer callData )
{
    //---- Start editable code block: MeasureBB acceptMeasured

    XmArrowButtonCallbackStruct *cbs = (XmArrowButtonCallbackStruct*) callData;

    //--- Comment out the following line when MeasureBB::acceptMeasured is implemented:
```

```
//////////////////////////////////////////////////////////////////////
//
// Source file for MeasureBB
//
//    This file is generated by RapidApp 1.2
//
//    This class is derived from MeasureBBUI which
//    implements the user interface created in
//    RapidApp. This class contains virtual
//    functions that are called from the user interface.
//
//    When you modify this source, limit your changes to
//    modifying the sections between the
//    "//---- Start/End editable code block" markers
//
//    This will allow RapidApp to integrate changes more easily
//
//    This class is a ViewKit user interface "component".
//    For more information on how components are used, see the
//    "ViewKit Programmers' Manual", and the RapidApp
//    User's Guide.
//////////////////////////////////////////////////////////////////////

#include "MeasureBB.h"
#include <Vk/VkEZ.h>
#include <Xm/ArrowB.h>
#include <Xm/BulletinB.h>
#include <Xm/Label.h>
#include <Xm/TextF.h>
#include <Vk/VkResource.h>


extern void VkUnimplemented ( Widget, const char * );

//////////////////////////////////////////////////////////////////////////////
// The following non-container elements are created by MeasureBBUI and are
// available as protected data members inherited by this class
//
//  XmLabel                 _labelImageDistance
//  XmLabel                 _labelImagePixel
//  XmLabel                 _labelVesselMeasured
//  XmLabel                 _labelVesselDefault
//  XmLabel                 _labelVessel
//  XmLabel                 _labelImage
//  XmArrowButton                   _arrow
//  XmTextField             _textfieldImageDistance
//  XmTextField             _textfieldImagePixel
//  XmTextField             _textfieldVesselMeasure
//  XmTextField             _textfieldVesselDistance
//  XmTextField             _textfiedVessel
//
//////////////////////////////////////////////////////////////////////////////


//---- Start editable code block: headers and declarations

#include "MagicBB.h"
#include <stdio.h>
#include <math.h>

//---- End editable code block: headers and declarations



//---- MeasureBB Constructor
```

```
  private:

    static void* RegisterMeasureBBInterface();

    //---- Start editable code block: MeasureBB private


    //---- End editable code block: MeasureBB private


};
//---- Start editable code block: End of generated code


//---- End editable code block: End of generated code

#endif
```

```
//////////////////////////////////////////////////////////////////////
//
// Header file for MeasureBB
//
//    This file is generated by RapidApp 1.2
//
//    This class is derived from MeasureBBUI which
//    implements the user interface created in
//    RapidApp. This class contains virtual
//    functions that are called from the user interface.
//
//    When you modify this header file, limit your changes to those
//    areas between the "//---- Start/End editable code block" markers
//
//    This will allow RapidApp to integrate changes more easily
//
//    This class is a ViewKit user interface "component".
//    For more information on how components are used, see the
//    "ViewKit Programmers' Manual", and the RapidApp
//    User's Guide.
//////////////////////////////////////////////////////////////////////
#ifndef MEASUREBB_H
#define MEASUREBB_H
#include "MeasureBBUI.h"
//---- Start editable code block: headers and declarations


//---- End editable code block: headers and declarations


//---- MeasureBB class declaration

class MeasureBB : public MeasureBBUI
{

  public:

    MeasureBB ( const char *, Widget );
    MeasureBB ( const char * );
    ~MeasureBB();
    const char *  className();

    static VkComponent *CreateMeasureBB( const char *name, Widget parent );

    //---- Start editable code block: MeasureBB public

    class VkComponent *_parent;
    void set(class VkComponent *v) {_parent = v;}

    //---- End editable code block: MeasureBB public



  protected:

    // These functions will be called as a result of callbacks
    // registered in MeasureBBUI

    virtual void acceptMeasured ( Widget, XtPointer );

    //---- Start editable code block: MeasureBB protected


    //---- End editable code block: MeasureBB protected
```

```
                XmNwidth, 50,
                XmNheig   40,
                (XtPoint  NULL );


    //---- Start editable code block: MeasureBBUI create


    //---- End editable code block: MeasureBBUI create
}

const char * MeasureBBUI::className()
{
    return ("MeasureBBUI");
}    // End className()



////////////////////////////////////////////////////////////////
// The following functions are static member functions used to
// interface with Motif.
////////////////////////////////////

void MeasureBBUI::acceptMeasuredCallback ( Widget    w,
                                           XtPointer clientData,
                                           XtPointer callData )
{
    MeasureBBUI* obj = ( MeasureBBUI * ) clientData;
    obj->acceptMeasured ( w, callData );
}



////////////////////////////////////////////////////////////////
// The following functions are called from the menu items
// in this window.
////////////////////////////////////

void MeasureBBUI::acceptMeasured ( Widget, XtPointer )
{
    // This virtual function is called from acceptMeasuredCallback.
    // This function is normally overriden by a derived class.

}



//---- Start editable code block: End of generated code


//---- End editable code block: End of generated code
```

```
                                XmNy, 170,
                                XmNwidth, 4
                                XmNheight, 4
                                (XtPointer) NULL );

_arrow = XtVaCreateManagedWidget  ( "arrow",
                                    xmArrowButtonWidgetClass,
                                    _baseWidget,
                                    XmNx, 289,
                                    XmNy, 105,
                                    XmNwidth, 60,
                                    XmNheight, 50,
                                    (XtPointer) NULL );

XtAddCallback ( _arrow,
                XmNactivateCallback,
                &MeasureBBUI::acceptMeasuredCallback,
                (XtPointer) this );

_textfieldImageDistance = XtVaCreateManagedWidget  ( "textfieldImageDistance",
                                                     xmTextFieldWidgetClass,
                                                     _baseWidget,
                                                     XmNx, 283,
                                                     XmNy, 171,
                                                     XmNwidth, 70,
                                                     XmNheight, 40,
                                                     (XtPointer) NULL );

_textfieldImagePixel = XtVaCreateManagedWidget  ( "textfieldImagePixel",
                                                  xmTextFieldWidgetClass,
                                                  _baseWidget,
                                                  XmNx, 152,
                                                  XmNy, 170,
                                                  XmNwidth, 70,
                                                  XmNheight, 40,
                                                  (XtPointer) NULL );

_textfieldVesselMeasure = XtVaCreateManagedWidget  ( "textfieldVesselMeasure",
                                                     xmTextFieldWidgetClass,
                                                     _baseWidget,
                                                     XmNx, 286,
                                                     XmNy, 50,
                                                     XmNwidth, 70,
                                                     XmNheight, 40,
                                                     (XtPointer) NULL );

_textfieldVesselDistance = XtVaCreateManagedWidget  ( "textfieldVesselDistance",
                                                      xmTextFieldWidgetClass,
                                                      _baseWidget,
                                                      XmNx, 150,
                                                      XmNy, 50,
                                                      XmNwidth, 70,
                                                      XmNheight, 40,
                                                      (XtPointer) NULL );

_textfiedVessel = XtVaCreateManagedWidget  ( "textfiedVessel",
                                             xmTextFieldWidgetClass,
                                             _baseWidget,
                                             XmNx, 30,
                                             XmNy, 49,
```

```
installDestroyHandler();

// Create widgets used in this component
// All variables are data members of this class

_labelImageDistance = XtVaCreateManagedWidget  ( "labelImageDistance",
                                                 xmLabelWidgetClass,
                                                 _baseWidget,
                                                 XmNlabelType, XmSTRING,
                                                 XmNx, 303,
                                                 XmNy, 220,
                                                 XmNwidth, 28,
                                                 XmNheight, 20,
                                                 (XtPointer) NULL );


_labelImagePixel = XtVaCreateManagedWidget  ( "labelImagePixel",
                                              xmLabelWidgetClass,
                                              _baseWidget,
                                              XmNlabelType, XmSTRING,
                                              XmNx, 163,
                                              XmNy, 220,
                                              XmNwidth, 36,
                                              XmNheight, 20,
                                              (XtPointer) NULL );


_labelVesselMeasured = XtVaCreateManagedWidget  ( "labelVesselMeasured",
                                                  xmLabelWidgetClass,
                                                  _baseWidget,
                                                  XmNlabelType, XmSTRING,
                                                  XmNx, 283,
                                                  XmNy, 19,
                                                  XmNwidth, 73,
                                                  XmNheight, 20,
                                                  (XtPointer) NULL );


_labelVesselDefault = XtVaCreateManagedWidget  ( "labelVesselDefault",
                                                 xmLabelWidgetClass,
                                                 _baseWidget,
                                                 XmNlabelType, XmSTRING,
                                                 XmNx, 158,
                                                 XmNy, 20,
                                                 XmNwidth, 53,
                                                 XmNheight, 20,
                                                 (XtPointer) NULL );


_labelVessel = XtVaCreateManagedWidget  ( "labelVessel",
                                          xmLabelWidgetClass,
                                          _baseWidget,
                                          XmNlabelType, XmSTRING,
                                          XmNx, 30,
                                          XmNy, 20,
                                          XmNwidth, 50,
                                          XmNheight, 20,
                                          (XtPointer) NULL );


_labelImage = XtVaCreateManagedWidget  ( "labelImage",
                                         xmLabelWidgetClass,
                                         _baseWidget,
                                         XmNlabelType, XmSTRING,
                                         XmNx, 30,
```

```
    //---- Start editable code block: MeasureBB constructor 2

    //---- End editable code block: MeasureBB constructor 2

}    // End Constructor


MeasureBBUI::MeasureBBUI ( const char *name, Widget parent ) : VkComponent ( name )
{
    //---- Start editable code block: MeasureBB pre-create

    //---- End editable code block: MeasureBB pre-create


    // Call creation function to build the widget tree.

     create ( parent );

    //---- Start editable code block: MeasureBB constructor

    //---- End editable code block: MeasureBB constructor

}    // End Constructor

MeasureBBUI::~MeasureBBUI()
{
    // Base class destroys widgets

    //---- Start editable code block: MeasureBBUI destructor

    //---- End editable code block: MeasureBBUI destructor
}    // End destructor


void MeasureBBUI::create ( Widget parent )
{
    Arg      args[6];
    Cardinal count;
    count = 0;

    // Load any class-defaulted resources for this object

    setDefaultResources ( parent, _defaultMeasureBBUIResources  );


    // Create an unmanaged widget as the top of the widget hierarchy

    _baseWidget = _measureBB = XtVaCreateWidget ( _name,
                                                  xmBulletinBoardWidgetClass,
                                                  parent,
                                                  XmNresizePolicy, XmRESIZE_GROW,
                                                  (XtPointer) NULL );

    // install a callback to guard against unexpected widget destruction
```

```
////////////////////////////////////////////////////////////////
//
// Source file for MeasureBBUI
//
//    This class implements the user interface created in
//    RapidApp.
//
//    Restrict changes to those sections between
//    the "//--- Start/End editable code block" markers
//
//    This will allow RapidApp to integrate changes more easily
//
//    This class is a ViewKit user interface "component".
//    For more information on how components are used, see the
//    "ViewKit Programmers' Manual", and the RapidApp
//    User's Guide.
//
//
////////////////////////////////////////////////////////////////


#include "MeasureBBUI.h" // Generated header file for this class

#include <Xm/ArrowB.h>
#include <Xm/BulletinB.h>
#include <Xm/Label.h>
#include <Xm/TextF.h>
#include <Vk/VkResource.h>
//---- Start editable code block: headers and declarations


//---- End editable code block: headers and declarations


// These are default resources for widgets in objects of this class
// All resources will be prepended by *<name> at instantiation,
// where <name> is the name of the specific instance, as well as the
// name of the baseWidget. These are only defaults, and may be overriden
// in a resource file by providing a more specific resource name

String  MeasureBBUI::_defaultMeasureBBUIResources[] = {
        "*labelImage.labelString:  Image",
        "*labelImageDistance.labelString:  mm",
        "*labelImagePixel.labelString:  pixel",
        "*labelVessel.labelString:  Vessel",
        "*labelVesselDefault.labelString:  Default",
        "*labelVesselMeasured.labelString:  Measured",
        "*tabLabel:  Measure",

        //---- Start editable code block: MeasureBBUI Default Resources


        //---- End editable code block: MeasureBBUI Default Resources

        (char*)NULL
};

MeasureBBUI::MeasureBBUI ( const char *name ) : VkComponent ( name )
{
    // No widgets are created by this constructor.
    // If an application creates a component using this constructor,
    // It must explictly call create at a later time.
    // This is mostly useful when adding pre-widget creation
    // code to a derived class constructor.
```

```
    Widget  _textfieldImageDistance;
    Widget  _textfieldImagePixel;
    Widget  _textfieldVesselDistance;
    Widget  _textfieldVesselMeasure;


    // These virtual functions are called from the private callbacks (below)
    // Intended to be overriden in derived classes to define actions

    virtual void acceptMeasured ( Widget, XtPointer );

    //---- Start editable code block: MeasureBB protected

    //---- End editable code block: MeasureBB protected


  private:

    // Array of default resources

    static String      _defaultMeasureBBUIResources[];


    // Callbacks to interface with Motif

    static void acceptMeasuredCallback ( Widget, XtPointer, XtPointer );

    //---- Start editable code block: MeasureBB private

    friend class MagicDeckTabbedDeck;
    friend class MagicBB;
    //---- End editable code block: MeasureBB private
};
//---- Start editable code block: End of generated code


//---- End editable code block: End of generated code

#endif
```

```
//////////////////////////////////////////////////////////////////
//
// Header file for MeasureBBUI
//
//    This file is generated by RapidApp 1.2
//
//    This class implements the user interface portion of a class
//    Normally it is not used directly.
//    Instead the subclass, MeasureBB is instantiated
//
//    To extend or alter the behavior of this class, you should
//    modify the MeasureBB files
//
//    Restrict changes to those sections between
//    the "//--- Start/End editable code block" markers
//
//    This will allow RapidApp to integrate changes more easily
//
//    This class is a ViewKit user interface "component".
//    For more information on how components are used, see the
//    "ViewKit Programmers' Manual", and the RapidApp
//    User's Guide.
//
//////////////////////////////////////////////////////////////////
#ifndef MEASUREBBUI_H
#define MEASUREBBUI_H
#include <Vk/VkComponent.h>


//---- Start editable code block: headers and declarations


//---- End editable code block: headers and declarations


class MeasureBBUI : public VkComponent
{

  public:

    MeasureBBUI ( const char *, Widget );
    MeasureBBUI ( const char * );
    ~MeasureBBUI();
    void create ( Widget );
    const char *  className();

    //---- Start editable code block: MeasureBB public

    //---- End editable code block: MeasureBB public


  protected:


    // Widgets created by this class

    Widget  _arrow;
    Widget  _labelImage;
    Widget  _labelImageDistance;
    Widget  _labelImagePixel;
    Widget  _labelVessel;
    Widget  _labelVesselDefault;
    Widget  _labelVesselMeasured;
    Widget  _measureBB;
    Widget  _textfiedVessel;
```

```
////////////////////////////////////////////////////////////////////////

// This file contains a single global function "VkUnimplemented".
// This function supports development using Fix+Continue.
// You can simply set a breakpoint in this function to stop in
// all unimplemented callback functions in a program.
////////////////////////////////////////////////////////////////////////

#include <iostream.h>

#include <Xm/Xm.h>

void VkUnimplemented(Widget w, const char *str)
{
    cerr << "The member function " << str << "()  was invoked";
    if ( w )
        cerr << " by " << XtName(w);
    cerr << endl << flush;
}
```

```
#include "ByteImage.h"

ByteImage::ByteImage()
{
  _image = new ImgBase <unsigned char>;
}

ByteImage::ByteImage(int w, int h, unsigned char **img)
{
  _image = new ImgBase <unsigned char>;
  _image -> set_width(w);
  _image -> set_height(h);
  _image -> set_imgdata(img);
}

ByteImage::~ByteImage()
{
  _image -> free_imgdata();
}

ByteImage *ByteImage::copy()
{
    int w = _image -> get_width();
    int h = _image -> get_height();
    unsigned char **p = _image -> copy_imgdata();

    ByteImage *img = new ByteImage(w, h, p);
    return img;
}
```

```
#ifndef BYTEIMAGE_H
#define BYTEIMAGE_H
#include "ImgBase.h"

class ByteImage
{
  public:
    ByteImage();
    ByteImage(int, int, unsigned char **);
    ~ByteImage();

    ByteImage *copy();

    ImgBase <unsigned char> *_image;

  protected:

};

#endif
```

```
int   w, h;
short **img;

if ((fpimage = fopen(filename,"rb"))!=NULL)
  {
    fseek(fpimage, (long) offset, (int) 0);
    fread(magicnum,sizeof(char),4,fpimage);
    fread(&header_length,sizeof(int),1,fpimage);

    fread(pxres,sizeof(int),1,fpimage);
    fread(pyres,sizeof(int),1,fpimage);
    fread(&bits_per_short,sizeof(int),1,fpimage);

    fseek(fpimage, (long) header_length, (int) 0);

    if ((simage =(short *)calloc((size_t)(*pxres)*(*pyres),sizeof(short)))==NULL)
    {
        printf("allocation failure for simage.\n");
        return NULL;
    }

    fread(simage, sizeof(short),(*pxres)*(*pyres), fpimage);
    fclose(fpimage);
  }
else
  return NULL;

    w = *pxres;
    h = *pyres;

    ImgBase <short> *aux = new ImgBase <short>;
    img = aux -> alloc_imgdata(h, h);
    if(img == NULL) return NULL;

for(i=0; i<h; i++)
{
  for(j=0; j<w; j++)
    img[i][j] = (short)((simage[i*w+j] - ImageOffset)/sclfctr);

  if(w < h)
  for(j=w; j<h; j++)
    img[i][j] = 0;
}

float m1, m2;
aux -> set_width(h);
aux -> set_height(h);
aux -> set_imgdata(img);
aux -> get_bound(&m1, &m2);
delete aux;

printf(" Min %f   Max %f  (%d  %d)\n", m1, m2, *pxres, *pyres);

*pxres = h;
*pyres = h;

free(simage);
return img;
}
```

```
    if(img == NULL) return FALSE;
    set_org(w, h, img);
  }
  else if(flag == 0 || flag == 1)
  {
    if(d < 0 || d > 4096) return FALSE;
    img =  read_pixels(fp, d, d);

    fclose(fp);

    if(img == NULL) return FALSE;
    set_org(d, d, img);
  }
  else if(flag == 3)
  {
    img =  readIrisBin(fp);
    fclose(fp);
    if(img == NULL) return FALSE;
    set_org(d, d, img);
  }

  return TRUE;
}

/*
 * Read in image file in a 1024x1024 IRIS bin data
 */
short **GEAngio::readIrisBin(FILE *fp)
{
    int  w;
    int  h;
    w = 1024;
    h = 1024;

    unsigned char *img = new unsigned char[w*h];

    fread(img, sizeof(unsigned char), w*h, fp);

    ImgBase <short> *aux = new ImgBase <short>;
    short **simg = aux -> alloc_imgdata(w, h);
    delete aux;
    if(simg == NULL) return NULL;

    int i, j;

    for(i=0; i<h; i++)
    for(j=0; j<w; j++)
      simg[i][j] = (short)img[i*w+j];

    delete img;
    return simg;
}

/*************************************************************************/
/* Read in image file and return a pointer to an image array, and */
/* number of xres and yres points */
short **GEAngio::readimage(int *pxres, int *pyres,
              char *filename, float sclfctr,float ImageOffset)
{
  int offset;
  short *simage;
  FILE *fpimage;
  int i,j;
  char magicnum[4];
  int header_length, bits_per_short;
  offset = 0;
```

```
#include "GEAngio.h"
#include "ImgBase.h"

#include <stdio.h>
#include <math.h>
#include <malloc.h>
#include <stdlib.h>

GEAngio::GEAngio() : MedImage()
{
}

GEAngio::~GEAngio()
{
}

Boolean GEAngio::open_file(int flag, char *fname)
{
  int d = 1024; // d = 512;
  FILE  *fp;
  char  cmd[400];

  if(flag == 1)
  {
    sprintf(cmd, "dcm_dump_element 0018 0015 %s DCM.anatomy > tmp", fname);
    system(cmd);
    if( (fp = fopen("DCM.anatomy", "r")) == NULL ) return FALSE;
    char *anatomy_tmp = new char[100];
    fgets(anatomy_tmp, 100, fp);
    fclose(fp);
    _anatomy = anatomy_tmp;
    printf(" anatomy ====> %s\n", _anatomy);
  }

  if( (fp = fopen(fname, "r")) == NULL ) return FALSE;

  if(flag == 0 || flag == 1)
  {
    fpos_t     filePosition;
    fpos_t     pixel_data_start;
    fpos_t     pixal_data_size = d*d*2;

    /*
    sprintf(cmd, "dcm_dump_element 0028 0010 %s DCM.rows > tmp", fname);
    system(cmd);
    sprintf(cmd, "dcm_dump_element 0028 0011 %s DCM.cols > tmp", fname);
    system(cmd);
    */

    fseek (fp, 0L, SEEK_SET);
    fgetpos(fp, &filePosition);
    fseek (fp, 0L, SEEK_END);
    fgetpos(fp, &filePosition);
    pixel_data_start = filePosition - pixal_data_size;
    fsetpos(fp, &pixel_data_start);
  }

  //int d = int(fsqrt(float(filePosition)/2.0));

  int w, h;
  short **img;

  if(flag == 2)
  {
    img =  readimage(&w, &h, fname, 1.0, 0.0);
    fclose(fp);
```

```
#ifndef GEANGIO_H
#define GEANGIO_H

#include "MedImage.h"
#include <Vk/VkComponent.h>

class GEAngio : public MedImage
{
  public:
    GEAngio();
    ~GEAngio();

    Boolean open_file(int, char *fname);

    short **readIrisBin(FILE *fp);
    short **readimage(int *pxres, int *pyres,
              char *filename, float sclfctr,float ImageOffset);

  protected:

};

#endif
```

```
    img[i] = &(img1[i * height]);
  return (img);
}

template < class LE >
LE *ImgBase <LE> :: alloc_1Ddata (int sz)
{
  LE   *data;

  if(!(data = (LE *)malloc(sz * sizeof(LE) ))) {
    printf ("Sorry, Computer stingy on memory (data) (%d)\n", sz);
    delete data;
    return NULL;
  }

  return data;
}

template < class LE >
void ImgBase <LE> :: free_imgdata ()
{
  delete *_imgdata;
  delete _imgdata;
}

template < class LE >
void ImgBase <LE>::get_bound(float *min_I,float *max_I)
{
    *min_I = 1.0e30;
    *max_I = -1.0e30;
    int    k;
    LE     *p;

    for(k=0, p=*_imgdata; k<(_width*_height); k++, p++) {
      if((float)(*p) < *min_I) {*min_I = *p;}
      if((float)(*p) > *max_I) {*max_I = *p;}
    }
}

template < class LE >
LE **ImgBase <LE> :: copy_imgdata ()
{
  LE **img = alloc_imgdata(_width, _height);
  for(int x=0; x<_width; x++)
  for(int y=0; y<_height; y++)
    img[x][y] = _imgdata[x][y];
  return img;
}
```

```
#include "ImgBase.h"
#include <stdio.h>
#include <malloc.h>

template < class LE >
ImgBase <LE> ::ImgBase()
{
  _imgdata = NULL;
}

template < class LE >
ImgBase <LE> ::ImgBase(int w, int h)
{
  _width = w;
  _height = h;
  _imgdata = alloc_imgdata(w, h);
}

template < class LE >
ImgBase <LE> ::ImgBase(int w, int h, LE **img)
{
  _width = w;
  _height = h;
  _imgdata = img;
}

template < class LE >
ImgBase <LE> ::~ImgBase()
{
  //if(_imgdata != NULL)  free_imgdata();
}

template < class LE >
void ImgBase <LE> ::init(int  w,  int h)
{
  _width = w;
  _height = h;
  _imgdata = alloc_imgdata(w, h);
}

template < class LE >
void ImgBase <LE> ::set_imgdata(LE **img)
{
  if (_imgdata != NULL) free_imgdata();
  _imgdata = img;
}

template < class LE >
LE **ImgBase <LE> :: alloc_imgdata (int width, int height)
{
   int  i;
   LE   **img, *img1;

   if( !(img = (LE **)malloc( (width * sizeof(img1))) ))
   {
     printf("Sorry, Computer getting stingy on memory (img) width =%d \n", width);
     return(NULL) ;
   }

   if(!(img1 = (LE *)malloc(width * height * sizeof(LE) ))) {
     printf ("Sorry, Computer stingy on memory (img1) (%d, %d)\n", width, height);
     free(img) ;
     return(NULL) ;
   }

   for (i=0; i<width; i++)
```

```
#ifndef IMGBASE_H
#define IMGBASE_H

template < class LE >
class ImgBase
{

  public:

    ImgBase ();
    ImgBase (int, int);
    ImgBase (int, int, LE **);
    ~ImgBase();

    void init(int, int);

    int get_width() {return _width;}
    int get_height() {return _height;}
    LE **get_imgdata() {return _imgdata;}

    void set_width(int w) {_width = w;}
    void set_height(int h) {_height = h;}
    void set_imgdata(LE **img);

    void get_bound(float *min_I,float *max_I);

    LE *alloc_1Ddata(int);
    LE **alloc_imgdata(int, int);
    void free_imgdata();
    LE **copy_imgdata();

    int    _width;
    int    _height;
    LE     **_imgdata;

  protected:

};

#endif
```

```
CC=         CC

CFLAGS=     -g -mips3 -n32

OBJECTS=    ImgBase.o MedImage.o GEAngio.o

TARGETS=    libImages.a

$(TARGETS): $(OBJECTS)
            ar ru $(TARGETS) $(OBJECTS)

print:      $(OBJECTS:.o=.C)
            lp -dLaserJet $(OBJECTS:.o=.C)
printh:     $(OBJECTS:.o=.h)
            lp -dLaserJet $(OBJECTS:.o=.h)

OBJ1=       ImgBase.o
$(OBJ1):    $(OBJ1:.o=.C)
            $(CC) -c $(CFLAGS) $(OBJ1:.o=.C)

OBJ2=       MedImage.o
$(OBJ2):    $(OBJ2:.o=.C)
            $(CC) -c $(CFLAGS) $(OBJ2:.o=.C)

OBJ3=       GEAngio.o
$(OBJ3):    $(OBJ3:.o=.C)
            $(CC) -c $(CFLAGS) $(OBJ3:.o=.C)
```

```
      min_sig = winMin;
      max_sig = winMax;
    }

    if(fabsf(max_sig - min_sig) < 1.e-10) return;

    for (i=0; i<_orgHeight; i++)
    for(j=0; j<_orgWidth; j++)
    {
        val = _orgImg[i][j];
        if (val <= min_sig) tmp = 0.0;
        else if(val >= max_sig) tmp = 255.0;
        else
          tmp = (val - min_sig)/(max_sig - min_sig) * 255.0;
        _grayImg[_orgHeight-i-1][j] = int(tmp);
    }
    printf("update_grayImg: %d %d\n", _orgWidth, _orgHeight);

}

short  **MedImage::read_pixels(FILE *fp, int w, int h)
{
    ImgBase <short> *aux = new ImgBase <short>;
    short **img = aux -> alloc_imgdata(w, h);
    delete aux;
    if(img == NULL) return NULL;

    unsigned short  pixel, p1, p2;
    unsigned char   pixel1, pixel2;
    short *p = *img;

    int  k = 0;
    while (!feof(fp) && k < w*h)
    {
      fread(&pixel1, 1, 1, fp);
      fread(&pixel2, 1, 1, fp);
      p1 = pixel1;
      p2 = pixel2;
      pixel = (p2<<8)|p1;

      *p = (short) (pixel);
      ++p;
      ++k;
    }

    //printf(" k = %d    w*h = %d    %f %f %f \n", k, w*h, (float)img[325][512],
    //  (float)img[461][526], (float)img[267][286]);

    if(k != w * h) return NULL;
    else return img;
}
```

```
#include "MedImage.h"

#include "ImgBase.h"
#include <stdio.h>
#include <math.h>

MedImage::MedImage()
{
  _orgImg = NULL;
  _grayImg = NULL;
  _anatomy = NULL;
}

MedImage::~MedImage()
{
  if(_grayImg != NULL) {delete *_grayImg, delete _grayImg;}
}

void MedImage::update_zoomImg(float z)
{
  if(_zoom != z) set_zoom(z);
  if(_zoom < 1.0)
  {
    _zoomWidth = int(_zoom * _orgWidth);
    _zoomHeight = int(_zoom * _orgHeight);
  }
}

void MedImage::change_grayImg(float dmin, float dmax)
{
    _winMin += dmin;
    _winMax += dmax;
    printf(" min %f    max  %f\n", _winMin, _winMax);
    update_grayImg(_winMin, _winMax);
}

void MedImage::init_grayImg()
{
    printf("init_grayImg: \n");

    ImgBase <unsigned char> *aux = new ImgBase <unsigned char>;
    _grayImg = aux -> alloc_imgdata(_orgWidth, _orgHeight);
    delete aux;
    if(_grayImg == NULL) return;

    printf("init_grayImg: %d %d\n", _orgWidth, _orgHeight);
}

void MedImage::update_grayImg(float winMin, float winMax)
{
    float            val, tmp;
    int              i, j;
    float            min_sig, max_sig;


    if(winMin != _winMin || winMax != _winMax)
      set_cw(winMin, winMax);

    if(winMin == winMax && winMax == 0)
    {
      ImgBase <short> *aux = new ImgBase <short> (_orgWidth, _orgHeight, _orgImg);
      aux -> get_bound(&min_sig,&max_sig);
      delete aux;
    }
    else
    {
```

```
#ifndef MEDIMAGE_H
#define MEDIMAGE_H

#include <stdio.h>

class MedImage
{
  public:
    MedImage();
    ~MedImage();

    char  *_anatomy;

    short  **_orgImg;
    int    _orgWidth, _orgHeight;

    float  _zoom;
    int    _zoomWidth, _zoomHeight;
    short  **_zoomImg;

    float  _winMin, _winMax;
    unsigned char **_grayImg;

    void set_org(int w, int h, short **o) {_orgWidth=w; _orgHeight=h; _orgImg=o;}
    void set_zoom(float z) {_zoom = z;}
    void set_cw(float c, float w) {_winMin=c; _winMax=w;}

    void init_grayImg();
    void update_zoomImg(float z);
    void update_grayImg(float c, float w);
    void change_grayImg(float dc, float dw);

    short  **read_pixels(FILE *fp, int w, int h);
  protected:

};

#endif
```

```
    // CanvasString operators
    char & operator [] ( int n );                          // Subscript
    CanvasString & operator = ( const CanvasString &rightCanvasString );   // Assignmer
    operator const char * () const;                        // Conversion

    // CanvasString relational operators
    int operator == ( const CanvasString &rightCanvasString );
    int operator != ( const CanvasString &rightCanvasString );
    int operator <  ( const CanvasString &rightCanvasString );
    int operator <= ( const CanvasString &rightCanvasString );
    int operator >  ( const CanvasString &rightCanvasString );
    int operator >= ( const CanvasString &rightCanvasString );
    int operator == ( char *rightSeq );
    int operator != ( char *rightSeq );
    int operator <  ( char *rightSeq );
    int operator <= ( char *rightSeq );
    int operator >  ( char *rightSeq );
    int operator >= ( char *rightSeq );

  private:

    // Data members
    int bufferSize;    // Size of the string buffer
    char *buffer,      // CanvasString buffer containing a null-terminated
                       //   sequence of characters
         nullChar;     // Used by the subscript operator for out of
                       //   bounds references

  // Friends (not member functions)

    // Input/output stream operators
    friend istream & operator >> ( istream &input,
                                   CanvasString &inputCanvasString );
    friend ostream & operator << ( ostream &output,
                                   const CanvasString &outputCanvasString );

};

#endif
```

```
//-----------------------------------------------------------------
//
//  CanvasString.h
//
//  Class declaration for the array implementation of the CanvasString ADT
//
//-----------------------------------------------------------------


#ifndef STRADT_H
#define STRADT_H

#include <iostream.h>
#include <stdio.h>

const int inputBufferLength = 256;  // Size of buffer used by >> op.

class CanvasString
{
  public:

    // Constructors
    CanvasString ();                                    // Default constructor
    CanvasString ( const char *charSeq );               // Initialize to char*
    CanvasString ( const CanvasString & valueCanvasString );   // Copy constructor

    // Destructor
    ~CanvasString ();

    // CanvasString input function (see >> and << operators below also)
    void readline ( istream &input, char delim = '\n' );
    void set(char *s);
    char *get() {return buffer;}
    void trim(char c);
    char *add(char c);
    void add(char c, int n);

    void add_front(char *s);
    void add_back(char *s);

    int equal(CanvasString s);
    int equal(char *s);

    int contain (char *s);

    void cut_front(char c);
    void cut_back(char c);
    void get_front(char c);

    char *get_Group();
    char *get_Patient();
    char *get_Data();
    char *get_Date();

    int  numOfChar(char c);

    int readToTag(FILE *fp, char *tag);
    int  empty();

    CanvasString *copy();

    // CanvasString functions
    void clear ();              // Clear string
    void deleteNth ( int n );   // Delete nth character
    int length () const;        // # characters in a string
    double toFloat () const;    // Converts string to floating point
```

```
    // Put the whitespace character back in the stream.
    if ( input.good() )
       input.putback(ch);

    // Return the state of the input stream.

    return input;
}

//---------------------------------------------------------------------

ostream & operator << ( ostream &output, const CanvasString &outputCanvasString )

// CanvasString output operation. Inserts outputCanvasString in ostream output.
// Returns the state of the output stream.

{
   output << outputCanvasString.buffer;
   return output;
}
```

```
//-----------------------------------------------------------------

int CanvasString:: operator >= ( char *rightSeq )

// "Greater than or equal to " relational operator. Returns 1 if
// a string is greater than or equal to rightSeq. Otherwise
// returns 0.

{
    return ( strcmp(buffer,rightSeq) >= 0 );
}

//-----------------------------------------------------------------
//
//  Friend functions
//
//-----------------------------------------------------------------

istream & operator >> ( istream &input, CanvasString &inputCanvasString )

// CanvasString input operation. Extracts a string from istream input and
// returns it in inputCanvasString. Returns the state of the input stream.

{
    char inputBuffer[inputBufferLength],  // Input buffer
         ch;                              // Input character
    int cnt;                              // Counts characters input

    // Skip leading whitespace characters (if any).

    ch = ' ';
    while ( input.good () &&
            ( ch == ' ' || ch == '\t' || ch == '\n' ) )
       input.get(ch);

    // Read in the string character-by-character until a whitespace
    // character is encountered or the end of the input buffer is
    // reached.

    cnt = 0;
    while ( input.good() &&
            ch != ' ' && ch != '\t' && ch != '\n' &&
            cnt < inputBufferLength-1                    )
    {
        inputBuffer[cnt++] = ch;
        input.get(ch);
    }

    // Append null character.

    inputBuffer[cnt++] = '\0';

    // Release old string buffer and allocate new one. Fill the new
    // buffer with the contents of the input buffer.

    delete [] inputCanvasString.buffer;            // Release old buffer
    inputCanvasString.bufferSize = cnt;            // Set size of new buffer
    inputCanvasString.buffer = new char [ cnt ];   // Allocate new buffer
    strcpy(inputCanvasString.buffer,inputBuffer);  // Copy into new buffer

    // If necessary, skip characters until whitespace encountered.

    while ( input.good() &&
            ch != ' ' && ch != '\t' && ch != '\n' )
        input.get(ch);
```

```
//-------------------------------------------------------------------

int CanvasString:: operator >= ( const CanvasString &rightCanvasString )

// "Greater than or equal to " relational operator. Returns 1 if
// a string is greater than or equal to rightCanvasString. Otherwise
// returns 0.

{
    return ( strcmp(buffer,rightCanvasString.buffer) >= 0 );
}

//-------------------------------------------------------------------

int CanvasString:: operator == ( char *rightSeq )

// Equality relational operator. Returns 1 if a string is equal to
// rightSeq. Otherwise returns 0.

{
    return ( strcmp(buffer,rightSeq) == 0 );
}

//-------------------------------------------------------------------

int CanvasString:: operator != ( char *rightSeq )

// Inequality relational operator. Returns 1 if a string is NOT
// equal to rightSeq. Otherwise returns 0.

{
    return ( strcmp(buffer,rightSeq) != 0 );
}

//-------------------------------------------------------------------

int CanvasString:: operator < ( char *rightSeq )

// "Less than" relational operator. Returns 1 if a string is less
// than rightSeq. Otherwise returns 0.

{
    return ( strcmp(buffer,rightSeq) < 0 );
}

//-------------------------------------------------------------------

int CanvasString:: operator > ( char *rightSeq )

// "Greater than" relational operator. Returns 1 if a string is
// greater than rightSeq. Otherwise returns 0.

{
    return ( strcmp(buffer,rightSeq) > 0 );
}
//-------------------------------------------------------------------

int CanvasString:: operator <= ( char *rightSeq )

// "Less than or equal to" relational operator. Returns 1 if
// a string is less or equal to than rightSeq. Otherwise
// returns 0.

{
    return ( strcmp(buffer,rightSeq) <= 0 );
}
```

```
//-----------------------------------------------------------------

CanvasString:: operator const char * () const

// Converts a string to a standard C (char*) string.

{
   return buffer;
}

//-----------------------------------------------------------------

int CanvasString:: operator == ( const CanvasString &rightCanvasString )

// Equality relational operator. Returns 1 if a string is equal to
// rightCanvasString. Otherwise returns 0.

{
    return ( strcmp(buffer,rightCanvasString.buffer) == 0 );
}

//-----------------------------------------------------------------

int CanvasString:: operator != ( const CanvasString &rightCanvasString )

// Inequality relational operator. Returns 1 if a string is NOT
// equal to rightCanvasString. Otherwise returns 0.

{
    return ( strcmp(buffer,rightCanvasString.buffer) != 0 );
}

//-----------------------------------------------------------------

int CanvasString:: operator < ( const CanvasString &rightCanvasString )

// "Less than" relational operator. Returns 1 if a string is less
// than rightCanvasString. Otherwise returns 0.

{
    return ( strcmp(buffer,rightCanvasString.buffer) < 0 );
}

//-----------------------------------------------------------------

int CanvasString:: operator > ( const CanvasString &rightCanvasString )

// "Greater than" relational operator. Returns 1 if a string is
// greater than rightCanvasString. Otherwise returns 0.

{
    return ( strcmp(buffer,rightCanvasString.buffer) > 0 );
}
//-----------------------------------------------------------------

int CanvasString:: operator <= ( const CanvasString &rightCanvasString )

// "Less than or equal to" relational operator. Returns 1 if
// a string is less or equal to than rightCanvasString. Otherwise
// returns 0.

{
    return ( strcmp(buffer,rightCanvasString.buffer) <= 0 );
}
```

```
        for ( j = n ; j <= length() ; j++ )     // Last shift is the
            buffer[j] = buffer[j+1];            // null character
}

//---------------------------------------------------------------------
int CanvasString::empty()
{
    //printf(" CanvasString::empty = %s %d\n", buffer, strlen(buffer));
    if(strlen(buffer) > 0) return 0;
    else return 1;
}

int CanvasString:: length () const

// Returns the number of characters in a string (excluding the
// null character).

{
    return strlen(buffer);
}

//---------------------------------------------------------------------

double CanvasString:: toFloat () const

// Converts a string to a floating-point number (double).

{
    return atof(buffer);
}


//---------------------------------------------------------------------

char & CanvasString:: operator [] ( int n )

// Returns the nth character in a string -- where the characters are
// numbered beginning with zero. If there is no nth character, then
// returns the null character.

{
    if ( n >= 0  &&  n < length() )
       return buffer[n];
    else
    {                       // Out of bounds reference
       nullChar = '\0';     // Restore nullChar (in case it was changed)
       return nullChar;
    }
}

//---------------------------------------------------------------------

CanvasString & CanvasString:: operator = ( const CanvasString &rightCanvasString )

// Assigns rightCanvasString to a string.

{
    if ( &rightCanvasString != this )       // Confirm non-trivial assignment
    {
       delete [] buffer;                        // Release buffer
       bufferSize = rightCanvasString.length()+1;
       buffer = new char [ bufferSize ];        // Allocate a new buffer
       strcpy(buffer,rightCanvasString.buffer);     // Copy rightCanvasString
    }
    return *this;
}
```

```
    buffer[0] = '\0';
    return;
  }

  for(i=n-1; i>=0; i--)
    if( buffer[i] != c ) {i1=i; break;}

  if(i0 > i1)
  {
    delete [] buffer;
    bufferSize = 1;
    buffer = new char [bufferSize];
    buffer[0] = '\0';
    return;
  }

  char *s = new char[bufferSize];
  strcpy(s, buffer);

  delete [] buffer;
  bufferSize = i1-i0+1;
  buffer = new char [bufferSize];

  for(i=i0; i<=i1; i++)
    buffer[i-i0] = s[i];
  buffer[i1-i0+1] = '\0';

  delete s;
}

char *CanvasString::add(char c)
{
  char *s = new char[bufferSize+1];

  for(int i=0; i<=(bufferSize-2); i++)
    s[i] = buffer[i];
  s[bufferSize-1] = c;
  s[bufferSize] = '\0';

  return s;
}

void CanvasString::add(char c, int n)
{
  char *s;
  while(bufferSize < n)
  {
    //cout << bufferSize << "buffer = {" << buffer << "}" << endl;
    s = add(c);
    set(s);
    //cout << "done" << endl;
  }
  //cout << "done" << endl;
}

//---------------------------------------------------------------------

void CanvasString:: deleteNth ( int n )

// Deletes the nth character in a string, where the characters are
// numbered beginning with zero.

{
    int j;   // Loop counter

    if ( n >= 0  &&  n < length() )
```

```
  n = strlen(buffer);
  if(n==0) return;

  i0 = -1;
  for(i=n-1; i>=0; i--)
    if( buffer[i] == c ) {i0 = i; break;}
  if(i0 <= 0) return;

  s2 = new char[i0];

  for(i=0; i<i0; i++)
    s2[i] = buffer[i];
  s2[i0] = '\0';
  set(s2);
}

//---------------------------------------------------------------------

void CanvasString:: clear ()

// Clears a string -- that is, makes it empty.

{
    buffer[0] ='\0';
}

int CanvasString::numOfChar(char c)
{
  int  i,k,n;

  n = strlen(buffer);
  if(n <= 0) return 0;

  k = 0;
  for(i=0; i<n; i++)
    if( buffer[i] == c ) k++;

  return k;
}

void CanvasString::trim(char c)
{
  int  i,i0,i1,n;

  n = strlen(buffer);

  if(n == 0)
  {
    delete [] buffer;
    bufferSize = 1;
    buffer = new char [bufferSize];
    buffer[0] = '\0';
    return;
  }

  i0 = -1;

  for(i=0; i<n; i++)
    if( buffer[i] != c ) {i0 = i; break;}

  if(i0 < 0)
  {
    delete [] buffer;
    bufferSize = 1;
    buffer = new char [bufferSize];
```

```
void CanvasString::add_front(char *s)
{
   char *s2 = new char[length()+strlen(s)];
   strcpy(s2, s);
   strcat(s2, buffer);
   set(s2);
}

void CanvasString::add_back(char *s)
{
   char *s2 = new char[length()+strlen(s)];
   strcpy(s2, buffer);
   strcat(s2, s);
   set(s2);
}

void CanvasString::cut_front(char c)
{
  int  i,i0,n;
  char *s2;

  n = strlen(buffer);
  if(n==0) return;

  i0 = -1;
  for(i=0; i<n; i++)
    if( buffer[i] == c ) {i0 = i; break;}

  if(i0 < 0) return;
  if(n-i0-1 <= 0) return;

  s2 = new char[n-i0-1];

  for(i=i0+1; i<n; i++)
    s2[i-i0-1] = buffer[i];
  s2[n-i0-1] = '\0';
  set(s2);
}

void CanvasString::get_front(char c)
{
  int  i,i0,n;
  char *s2;

  n = strlen(buffer);
  if(n==0) return;

  i0 = -1;
  for(i=0; i<n; i++)
    if( buffer[i] == c ) {i0 = i; break;}

  if(i0 <= 0) return;

  s2 = new char[i0];

  for(i=0; i<i0; i++)
    s2[i] = buffer[i];
  s2[i0] = '\0';
  set(s2);
}


void CanvasString::cut_back(char c)
{
  int  i,i0,n;
  char *s2;
```

```
}

char *CanvasString::get_Group()
{
    CanvasString *cs = copy();
    cs -> cut_front('@');
    cs -> get_front('@');
    return cs -> get();
}

char *CanvasString::get_Patient()
{
    CanvasString *cs = copy();
    cs -> cut_front('@');
    cs -> cut_front('@');
    cs -> get_front('@');
    return cs -> get();
}

char *CanvasString::get_Data()
{
    CanvasString *cs = copy();
    cs -> cut_front('@');
    cs -> cut_front('@');
    cs -> cut_front('@');
    cs -> get_front('@');
    return cs -> get();
}

char *CanvasString::get_Date()
{
    CanvasString *cs = copy();
    cs -> cut_front('@');
    cs -> cut_front('@');
    cs -> cut_front('@');
    cs -> cut_front('@');
    cs -> get_front('@');
    return cs -> get();
}

void CanvasString::set(char *s)
{
     delete [] buffer;
     bufferSize =  strlen(s) + 1;
     buffer = new char[bufferSize];
     strcpy(buffer,s);
}

int CanvasString::equal(CanvasString s)
{
     if(strcmp(buffer, s.get()) == 0) return 1;
     else return 0;
}

int CanvasString::equal(char *s)
{
     if(strcmp(buffer, s) == 0) return 1;
     else return 0;
}

int CanvasString::contain (char *s)
{
     if( strstr(buffer,s) == NULL ) return 0;
     else return 1;
}
```

```
int CanvasString::readToTag(FILE *fp, char *tag)
{
    char  s[300];
    int   flag;
    CanvasString *cs = new CanvasString();

    set("");

    if(feof(fp)) return 0;

    fgets(s, 300, fp);
    //printf(" ||%s \n", s);
    if(strstr(s,tag) == NULL)
    {
      flag = 1;
      set(s);
      trim('\t');
      trim('\n');
      trim(' ');
    }
    else flag = 0;

    while(flag)
    {
      if(feof(fp)) return 0;

      fgets(s, 300, fp);
      //printf(" ||%s \n", s);
      if(strstr(s,tag) == NULL)
      {
          flag = 1;
          add_back("\n");
          cs -> set(s);
          trim('\t');
          trim('\n');
          cs -> trim(' ');
          add_back(cs -> get());
      }
      else flag = 0;
    };

    return 1;
}

void CanvasString:: readline ( istream &input, char delim )

// CanvasString input function. Reads characters from istream input until
// the delim character is read in (retains up to inputBufferLength-1
// characters).

{
    char inputBuffer[inputBufferLength];  // Input buffer
    int cnt;                              // Size of input string

    // Read in the string and set cnt to its length.

    input.getline(inputBuffer,inputBufferLength,delim);
    cnt = strlen(inputBuffer) + 1;

    // Release the old string buffer and allocate a new one. Fill the
    // new buffer with the contents of the input buffer.

    delete [] buffer;             // Release old buffer
    bufferSize = cnt;             // Set size of new buffer
    buffer = new char [ cnt ];    // Allocate new buffer
    strcpy(buffer,inputBuffer);   // Copy into new buffer
```

```
//---------------------------------------------------------------------
//
//  Array implementation of the String ADT
//
//---------------------------------------------------------------------

#include <iostream.h>
#include <string.h>
#include <stdio.h>
#include <math.h>       // For the atof() function
#include "CanvasString.h"

//---------------------------------------------------------------------

CanvasString:: CanvasString ()

// Constructor. Creates an empty string.

{
    bufferSize = 1;                    // Store the buffer size
    buffer = new char [bufferSize];    // Allocate the buffer
    buffer[0] = '\0';                  // Initialize to empty string
}

//---------------------------------------------------------------------

CanvasString:: CanvasString ( const char *charSeq )

// Constructor. Creates a string containing the delimited sequence of
// characters charSeq.

{
    bufferSize = strlen(charSeq)+1;    // Store the buffer size
    buffer = new char [bufferSize];    // Allocate the buffer
    strcpy(buffer,charSeq);            // Copy charSeq into the buffer
}

//---------------------------------------------------------------------

CanvasString:: CanvasString ( const CanvasString & valueCanvasString )

// Copy constructor. Creates a string containing valueCanvasString.

{
    bufferSize = valueCanvasString.length()+1;    // Store the buffer size
    buffer = new char [bufferSize];          // Allocate the buffer
    strcpy(buffer,valueCanvasString.buffer);     // Copy valueCanvasString
}

//---------------------------------------------------------------------

CanvasString:: ~CanvasString ()

// Destructor. Deallocates the string buffer.

{
    delete [] buffer;    // Deallocate the string buffer
}

//---------------------------------------------------------------------
CanvasString *CanvasString::copy()
{
    CanvasString *cs = new CanvasString(get());
    return cs;
}
```

```
    // List iteration operations
    int gotoBeginning ();                    // Go to beginning
    int gotoEnd ();                          // Go to end
    int gotoNext ();                         // Go to next element
    int gotoPrior ();                        // Go to prior element
    LE getCursor () const;                   // Return element

    // Output the list structure -- used in testing/debugging
    void showStructure () const;

    // In-lab operations
    void moveToBeginning ();                 // Move to beginning
    void insertBefore ( const LE &newElement );
                                             // Insert before cursor

  private:

    // Data members
    ListNode<LE> *head,      // Pointer to the beginning of the list
                 *cursor;    // Cursor pointer
};

# endif
```

```
//-----------------------------------------------------------------
//
//                  list.h
//
//  Class declarations for the linked list implementation of the
//  List ADT
//
//-----------------------------------------------------------------
# ifndef  LIST_H
# define  LIST_H

#include <stdio.h>
#include "CanvasString.h"

template < class LE >
class List;

template < class LE >
class ListNode                  // Facilitator class for the List class
{
  private:

    // Constructor
    ListNode ( const LE &elem, ListNode *nextPtr );

    // Data members
    LE element;        // List element
    ListNode <LE> *next;    // Pointer to the next element

  friend class List<LE>;
};

//-----------------------------------------------------------------

template < class LE >
class List
{
  public:

    // Constructor
    List ( int ignored = 0 );

    // Destructor
    ~List ();

    int length();

    // List manipulation operations
    void insert ( const LE &newElement );     // Insert after cursor
    void remove ();                           // Remove element
    void replace ( const LE &newElement );    // Replace element
    void clear ();                            // Clear list
    int  member (const LE &newElement);
    int  member (char *key);

    void print();
    void saveToFile(FILE *fp);
    void openFile(FILE *fp, CanvasString *);

    LE   *retrieve(char *key);
    LE   *retrieve(const LE &newElement);

    // List status operations
    int empty () const;                       // List is empty
    int full () const;                        // List is full
```

```
    if (  head != 0 )
        {           sor->next = new ListNode <LE>(  cursor -> element, cursor->
             curso  -> element = newElement;
             cursor = cursor -> next;
        }
        else
        {
        head = new ListNode <LE> ( newElement,   0 ) ;
        cursor = head ;
        }


 }

//-------------------------------------------------------------------------
```

```
      for ( p = head ; p != 0 ; p = p->next )
         p->element.saveFile(fp);
   }
}

template < class LE >
void List<LE>:: openFile (FILE *fp, CanvasString *s)
{
    ListNode<LE> *p;

    if ( head == 0 )
       return;
    else
    {
       for ( p = head ; p != 0 ; p = p->next )
          p->element.openFile(fp, s);
    }
}

//---------------------------------------------------------------------------

    // In-lab operations
     template < class LE >

  void List < LE > :: moveToBeginning ()                  // Move to beginning

        {
           LE  old = cursor ->element;;

           remove(  );//have to one by one to find the element will to be delete.
           head = new ListNode <LE> (old, head);
           cursor = head;

        }

//---------------------------------------------------------------------------

 /*    template < class LE >

  void List < LE > ::  insertBefore ( const LE &newElement ) // Insert before cursor

         {
           ListNode < LE >*p = new ListNode < LE > ( newElement, cursor);
           ListNode < LE > *q = head;
            if ( cursor != head )
                {       while ( q -> next != cursor )
                                q = q->next;
                        q ->next = p;
                        cursor = p;
                }
                else
                {
                head = p;
                cursor = p ;
                }


         }
*/
//---------------------------------------------------------------------------
    template < class LE >

  void List < LE > ::  insertBefore ( const LE &newElement ) // Insert before cursor

         {
```

```
      return NULL;
   }
   else
   {
      for (p = head; p != 0; p = p->next)
      {
        cursor = p;
        if(p -> element.equal(newElement)) return &(p -> element);
      }
      return NULL;
   }
}

//---------------------------------------------------------------------------

   // Output the list structure -- used in testing/debugging

template < class LE >
void List<LE>:: showStructure () const

// Outputs the elements in a list. If the list is empty, outputs
// "Empty list". This operation is intended for testing and
// debugging purposes only.

{
/*
   ListNode<LE> *p;   // Iterates through the list

   if ( head == 0 )
      //cout << "Empty list" << endl;
   else
   {
      for ( p = head ; p != 0 ; p = p->next )
          if ( p == cursor )
             //cout << "[" << p->element << "] ";
          else
             //cout << p ->element << " ";
      //cout << endl;
   }
*/
}

template < class LE >
void List<LE>:: print ()
{
   ListNode<LE> *p;

   if ( head == 0 )
      printf("Empty list \n");
   else
   {
      for ( p = head ; p != 0 ; p = p->next )
         p->element.print();
      printf("\n");
   }
}

template < class LE >
void List<LE>:: saveToFile (FILE *fp)
{
   ListNode<LE> *p;

   if ( head == 0 )
      return;
   else
   {
```

```
int  List < LE > :: gotoNext ()              // Go to next element
{   assert( head !=0 );
        if ( cursor -> next != 0 )
        {     cursor = cursor -> next;
       return 1;
        }
        else
                return 0;
}

//---------------------------------------------------------------------

template < class LE >

int  List < LE > :: gotoPrior ()                          // Go to prior element
{
        ListNode <LE>  *p = head;
    assert(! empty() );
        if ( cursor == head )
                return 0;
        else
        {
               while ( p -> next != cursor )
        p = p -> next ;
                cursor = p;
        }
         return 1;
}
//---------------------------------------------------------------------

  template < class LE >

LE  List < LE > :: getCursor () const                     // Return element
  {

   LE temp;
       temp = cursor -> element;
       return temp;
  }


template < class LE >
LE  *List < LE > :: retrieve (char *key)
{
    ListNode<LE> *p;

    if ( head == 0 )
    {
       return NULL;
    }
    else
    {
       for (p = head; p != 0; p = p->next)
         if(p -> element.equal(key)) return &(p -> element);
       return NULL;
    }
}

template < class LE >
LE  *List < LE > :: retrieve (const LE &newElement)
{
    ListNode<LE> *p;

    if ( head == 0 )
    {
```

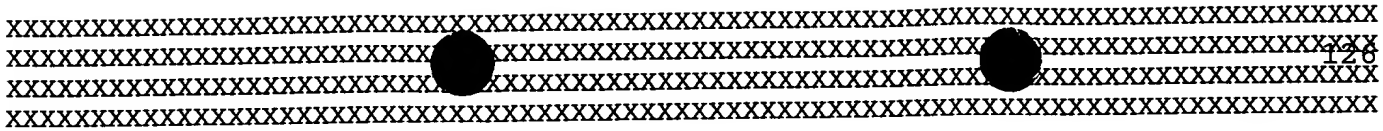
```
                p = 1;
        }
        else
                temp = 0;

   return temp;
        }

//---------------------------------------------------------------------------
template < class LE >
int  List < LE > :: length ()
{
    ListNode<LE> *p;
    int  length = 0;

    if ( head == 0 )
       return 0;
    else
    {
       for (p = head; p != 0; p = p->next)
         length ++;
        return length;
    }
}

//---------------------------------------------------------------------------
template < class LE >
int  List < LE > :: member (const LE &newElement)
{
    ListNode<LE> *p;

    //printf("       List < LE > :: member \n");

    if ( head == 0 )
       return 0;
    else
    {
       for (p = head; p != 0; p = p->next)
         if(p -> element.equal(newElement)) return 1;
        return 0;
    }
}

template < class LE >
int  List < LE > :: member (char *key)
{
    ListNode<LE> *p;

    if ( head == 0 )
    {
       printf(" List < LE > :: member (char *key)  head= 0\n");
       return 0;
    }
    else
    {
       for (p = head; p != 0; p = p->next)
         if(p -> element.equal(key)) return 1;
        return 0;
    }
}

//---------------------------------------------------------------------------

  template < class LE >
```

```
        }
//-------------------------------------------------------------------------

    // List status operations
    template < class LE >

  int  List < LE > :: empty () const                          // List is empty
        {
        if ( head == 0 )
                return 1;
        else
                return 0 ;

        }

//-------------------------------------------------------------------------

     template < class LE >

  int  List < LE > :: full () const                           // List is full
         {
           return 0;
         }

//-------------------------------------------------------------------------
    // List iteration operations
   template < class LE >

  int  List < LE > :: gotoBeginning ()                        // Go to beginning
   {
           if (! empty() )
           {
                   cursor = head;
           return 1;
           }
           else
                   return 0;
   }

//-------------------------------------------------------------------------
/* template < class LE >

  int  List < LE > :: gotoEnd ()                        // Go to  end
   {
           if (! empty() )
           {
                  while( cursor->next != 0 )
                          cursor = cursor -> next;
           return 1;
           }
           else
                   return 0;
   }
 */
//-------------------------------------------------------------------------
     template < class LE >

  int  List < LE > :: gotoEnd ()                              // Go to end

        {
        int temp;
         if ( ! empty() )
         { for (   cursor;   cursor-> next ;  cursor = cursor -> next )// not all
                                 // control paths return a value
```

```
        assert( head != 0 );
        cursor = head;

   }
   }

//---------------------------------------------------------------------

     template < class LE >
void   List < LE > :: remove ()                          // Remove element
         {


         assert( !empty() );
      if ( head -> next == 0)//one element;
                {   delete cursor;
                        head = 0;
                    cursor = 0;
                       }
          else
          {
                 ListNode <LE> *temp= head;
           if ( cursor == head )
          { cursor = cursor->next;
            head = cursor;
                delete temp;
          }
          else
          {
                ListNode <LE> *p= cursor;
          while (  temp-> next != cursor )
              temp = temp -> next;
                  temp -> next = cursor -> next;
                  if ( cursor->next ==0)
                          cursor = head;
                  else
              cursor = cursor -> next;
                   delete p;
          }
         }

         }

//---------------------------------------------------------------------

          template < class LE >

void   List < LE > ::  replace ( const LE &newElement )   // Replace element
          {
             if ( ! empty () )
                          cursor -> element = newElement;



          }

//---------------------------------------------------------------------

          template < class LE >

void   List < LE > :: clear ()                           // Clear list
          {
          head = 0;
          cursor = 0;
```

```
//-------------------------------------------------------------------
//
//     listlnk.C
//
//  Class declarations for the linked list implementation of the
//  List ADT
//
//-------------------------------------------------------------------


#include "Listlnk.h"
#include <iostream.h>
#include <assert.h>

template <class LE >
class List;

template < class LE > // Facilitator class for the List class

ListNode  <LE> :: ListNode ( const LE &elem, ListNode *nextPtr )// Constructor
                    :element ( elem ), next ( nextPtr )

{}



//-------------------------------------------------------------------

template < class LE >
  List < LE > :: List ( int ignored )// = 0 )// Constructor
                   : head( 0 ), cursor ( 0 )
{}

//-------------------------------------------------------------------

   template < class LE >
  List < LE > :: ~List ()// Destructor
   {
     clear();

   }

//-------------------------------------------------------------------

    // List manipulation operations

   template < class LE >
   void   List < LE > :: insert ( const LE &newElement )    // Insert after cursor
   {

   //printf(" List::insert start \n");

   if( !empty() )
   {
     //printf(" List::!empty()\n");
     cursor -> next = new ListNode < LE > ( newElement, cursor -> next );
     assert( (cursor -> next)!= 0);
     cursor = cursor -> next;

   }

   else
   {
     //printf(" List::empty()\n");
     head = new ListNode < LE > ( newElement, 0 );
```

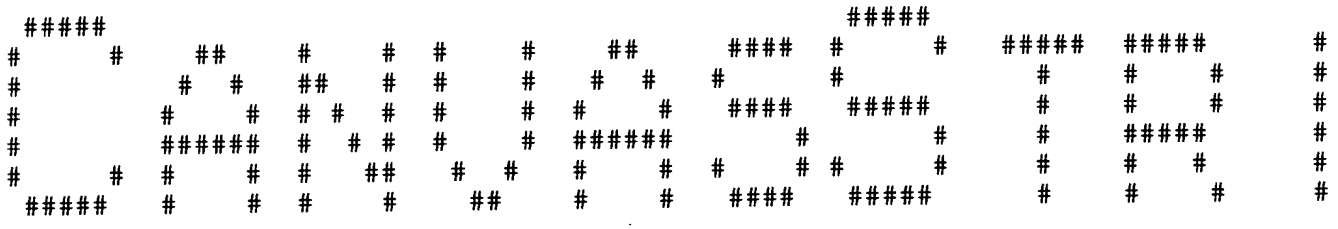
```
CC=         CC

CFLAGS=     -g -mips3 -n32

OBJECTS=    CanvasString.o Listlnk.o

TARGETS=    libGeneral.a

$(TARGETS): $(OBJECTS)
            ar ru $(TARGETS) $(OBJECTS)

OBJ1=       CanvasString.o
$(OBJ1):    CanvasString.h $(OBJ1:.o=.C)
            $(CC) -c $(CFLAGS) $(OBJ1:.o=.C)

OBJ2=       Listlnk.o
$(OBJ2):    Listlnk.h $(OBJ2:.o=.C)
            $(CC) -c $(CFLAGS) $(OBJ2:.o=.C)
```

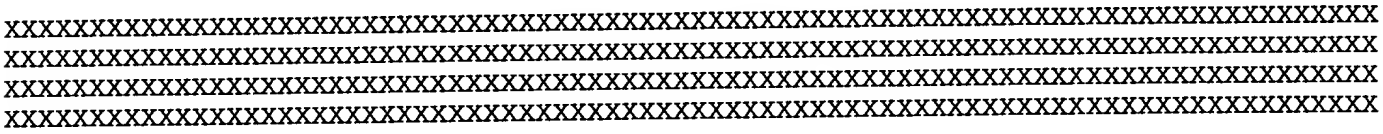


User: Meide Zhao

Request id: DeskJet2-700 Printer: DeskJet2

Fri Sep 10 12:38:14 CDT 1999





```
#ifndef DRAW_H
#define DRAW_H

/*
 * File:         Draw.h
 *
 * Author:       Ying Dai
 *               ydai@eecs.uic.edu
 *
 * Description:
 *       This file contains the declaration of X window variables.
 */

#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/Intrinsic.h>

class Draw
{
public:
    static Display *display;
    static Window window;
    static int screen;
    static GC bgGC;
    static GC redGC;
    static GC whiteGC;
    static GC greenGC;
    static GC yellowGC;
    static GC blueGC;
    static GC blackGC;
    static unsigned long fg;
    static unsigned long bg;

    static void init(Widget canvas);
    static void redisplay(Widget canvas, XtPointer, XtPointer);
};

#endif // DRAW_H
```

```
                XmNcolormap, &colormap,
                XmNforeground, &redGC,
                XmNbackground, &bg,
                XmNdepth,    &depth,
                NULL);
    XAllocNamedColor(dpy, cmap, "black", &color, &ignore);
    bgColor = color.pixel;
    XtVaSetValues(canvas, XmNbackground, bgColor, NULL);

    values.foreground = Red;
    values.background = bgColor;
    redGC = XtAllocateGC( canvas, depth, GCForeground|GCBackground, &values,
                GCFont, 0);
    values.foreground = values.background;
    bgGC = XtGetGC( canvas, GCForeground|GCBackground, &values);

    XAllocNamedColor(display, colormap,  "black", &color, &ignore);
    Black = color.pixel;
    values.foreground = Black;
    blackGC = XtGetGC(canvas, GCForeground | GCBackground, &values);

    XAllocNamedColor(display, colormap,  "red", &color, &ignore);
    Red = color.pixel;
    values.foreground = Red;
    redGC = XtGetGC(canvas, GCForeground | GCBackground, &values);

    XAllocNamedColor(display, colormap,  "white", &color, &ignore);
    White = color.pixel;
    values.foreground = White;
    whiteGC = XtGetGC(canvas, GCForeground | GCBackground, &values);

    XAllocNamedColor(display, colormap,  "yellow", &color, &ignore);
    Yellow = color.pixel;
    values.foreground = Yellow;
    yellowGC = XtGetGC(canvas, GCForeground | GCBackground, &values);

    XAllocNamedColor(display, colormap,  "blue", &color, &ignore);
    Blue = color.pixel;
    values.foreground = Blue;
    blueGC = XtGetGC(canvas, GCForeground | GCBackground, &values);

    XAllocNamedColor(display, colormap,  "green", &color, &ignore);
    Green = color.pixel;
    values.foreground = Green;
    greenGC = XtGetGC(canvas, GCForeground | GCBackground, &values);
}


/*
 * Name:        Draw::redisplay
 *
 * Description:
 *      Redraw the image.
 *
 * Parameters:
 *      Widget canvas
 *      XtPointer
 *      XtPointer
 *
 * Return value:
 *      None
 */

    void
Draw::redisplay(Widget canvas, XtPointer, XtPointer)
{
```

```
/*
 * File:        Draw.C
 *
 * Author:      Ying Dai
 *              ydai@eecs.uic.edu
 *
 * Description:
 *      This file contains the implementation of the Draw calss.
 */

#include <Xm/Xm.h>
#include <stdio.h>
#include "Draw.h"
#include "Vessel.h"

// Instantiate static data members.
Display *Draw::display;
Window Draw::window;
int Draw::screen;
unsigned long Draw::fg;
unsigned long Draw::bg;
GC Draw::redGC;
GC Draw::yellowGC;
GC Draw::blueGC;
GC Draw::greenGC;
GC Draw::whiteGC;
GC Draw::blackGC;
GC Draw::bgGC;
Pixel bgColor;
Pixel Black;
Pixel Red;
Pixel Blue;
Pixel Yellow;
Pixel Green;
Pixel White;
Colormap colormap;

/*
 * Name:        Draw::init
 *
 * Description:
 *      This function initializes the X window variables.
 *
 * Parameters:
 *      Widget canvas
 *
 * Return value:
 *      None
 */

    void
Draw::init(Widget canvas)
{
    Display *dpy = XtDisplay(canvas);
    int depth;
    int scr = DefaultScreen(dpy);
    Colormap cmap = DefaultColormap(dpy, scr);
    XGCValues values;
    XColor color, ignore;
    display = XtDisplay(canvas);
    screen = XDefaultScreen(display);
    window = XtWindow(canvas);
    //bg = 1;     //WhitePixel(display, screen);
    //fg = 0;     //BlackPixel(display, screen);

    XtVaGetValues(canvas,
```

```
    virtual void motion ( Widget, XEvent * );

    void clear_memory();
    void init_OpenGL();

  //---- End editable code block: ModelDraw protected


private:

  static void* RegisterModelDrawInterface();

  //---- Start editable code block: ModelDraw private

    int             _init_OpenGL;
    int             _width,_height;
    unsigned char   **_grayImg, **_gray;

  //---- End editable code block: ModelDraw private


};
//---- Start editable code block: End of generated code


//---- End editable code block: End of generated code

#endif
```

```
#ifndef MODELDRAW_H
#define MODELDRAW_H
#include "DrawingAreaUI.h"
//---- Start editable code block: headers and declarations

#include <GL/GLwMDrawA.h>

//---- End editable code block: headers and declarations

int    MODEL_WIDTH,MODEL_HEIGHT;

float           std_diameter[108];
float           meas_diameter[108];

//---- ModelDraw class declaration

class ModelDraw : public DrawingAreaUI
{

  public:

     ModelDraw(int x, int y, int w, int h, const char *name, Widget parent);

     ~ModelDraw();
     const char *  className();

    //---- Start editable code block: ModelDraw public

     int get_width() {return _width;}
     int get_height() {return _height;}
     unsigned char **get_image() {return _grayImg;}

     void init();
     void set(int w, int h) {_width = w; _height = h;}
     void set(unsigned char **grayimg);
     void set(int w,int h, unsigned char **grayimg);

     void init_display();
     void clear_display();
     void display();
     void display(int, int);

     class VkComponent *_parent;
     void set(class VkComponent *p) {_parent = p;}

     GLXContext _glxContext;

     void set_measured(int v, float d);

    //---- End editable code block: ModelDraw public


  protected:

    // These functions will be called as a result of callbacks
    // registered in ModelDrawUI

    virtual void ginit ( Widget, XtPointer );
    virtual void expose ( Widget, XtPointer );
    virtual void input ( Widget, XtPointer );
    virtual void resize ( Widget, XtPointer );

    //---- Start editable code block: ModelDraw protected
```

```
    h = _height;

    _grayImg = img -> allo_imgdata(w, h);

    for(x=0; x<w; x++)
    for(y=0; y<h; y++)
    {
      _grayImg[x][y] = 0;
    }
}

void ModelDraw::set_measured(int v, float d)
{
   meas_diameter[v] = d;

   FILE *fp;

   if( (fp = fopen(_saveFile, "w")) == NULL )
   {
       printf(" can't open file %s \n", _saveFile);
       return;
   }
   else printf(" Write to File %s\n", _saveFile);

   for(int i=0; i<108; i++)
   {
     fprintf(fp, "%d  %10.5f\n", i+1, meas_diameter[i]);
   }
   fclose(fp);

   //printf(" meas_diameter:: %d   %f\n", v, d);
}

//---- End editable code block: End of generated code
```

```
    static InterfaceMap map[] = {
    //---- Start editable code block: ModelDrawUI resource table

      // { "resourceName", "setAttribute", XmRString},
    //---- End editable code block: ModelDrawUI resource table
      { NULL }, // MUST be NULL terminated
    };

    return map;
} // End RegisterModelDrawInterface()




//---- End of generated code

//---- Start editable code block: End of generated code

void ModelDraw::init()
{
    _init_OpenGL = 0;

    FILE  *fp;

    int  i, k;
    for(i=0; i<108; i++)
    {
        meas_diameter[i] = 0.0;
    }

    if( (fp=fopen(".curr_patient", "r")) == NULL )
    {
      return;
    }
    char filename[300];
    fscanf(fp, "%s", filename);
    fclose(fp);

    sprintf(_saveFile, "measured.dat.%s", filename);
    if( (fp = fopen(_saveFile, "r")) != NULL)
    {
      for(i=0; i<108; i++)
      {
        fscanf(fp, "%d  %f", &k, &meas_diameter[i]);
      }
      fclose(fp);
    }
    else if( (fp = fopen("measured.dat.std", "r")) != NULL)
    {
      for(i=0; i<108; i++)
      {
        fscanf(fp, "%d  %f", &k, &meas_diameter[i]);
      }
      fclose(fp);
    }

    ImgBase <unsigned char> *img = new ImgBase <unsigned char>;
    int w, h, x, y;
    float  tmp;
    _width =  get_widthUI();
    _height = get_heightUI();
    w = _width;
```

```
void ModelDraw::motion ( Widget w, XEvent *event )
{
    //---- Start editable code block: ModelDraw resize

    //XmDrawingAreaCallbackStruct *cbs = (XmDrawingAreaCallbackStruct*) callData;

    //--- Comment out the following line when ModelDraw::resize is implemented:

    //::VkUnimplemented ( w, "ModelDraw::resize" );

    int xpos = event->xmotion.x;
    int ypos = event->xmotion.y;
     //printf(" Motion \n");
    //---- End editable code block: ModelDraw resize

}    // End ModelDraw::resize()



////////////////////////////////////////////////////////////////////
// static creation function, for importing class into rapidapp
// or dynamically loading, using VkComponent::loadComponent
////////////////////////////////////////////////////////////////////


////////////////////////////////////////////////////////////////////
// Function for accessing a description of the dynamic interface
// to this class.
////////////////////////////////////////////////////////////////////


// WARNING: This structure is different than that used with 1.1 RapidApp.
// See the RapidApp release notes for details

struct InterfaceMap {
  char  *resourceName;
  char  *methodName;
  char  *argType;
  char  *definingClass; // Optional, if not this class
  void (VkCallbackObject::*method)(...); // Reserved, do not set
};


void *ModelDraw::RegisterModelDrawInterface()
{
    // This structure registers information about this class
    // that allows RapidApp to create and manipulate an instance.
    // Each entry provides a resource name that will appear in the
    // resource manager palette when an instance of this class is
    // selected, the name of the member function as a string,
    // the type of the single argument to this function, and an.
    // optional argument indicating the class that defines this function.
    // All member functions must have the form
    //
    //     void memberFunction ( Type );
    //
    // where "Type" is one of:
    //    const char *     (Use XmRString)
    //    Boolean          (Use XmRBoolean)
    //    int              (Use XmRInt)
    //    float            (Use XmRFloat)
    //    No argument      (Use VkRNoArg or "NoArg"
    //    A filename       (Use VkRFilename or "Filename")
    //    An enumeration   (Use "Enumeration:ClassName:Type: VALUE1, VALUE2, VALUE3")
    //    A callback       (Use XmRCallback)
```

```
    if (cb->event->type == ButtonPress)
    {
        if (cb->event->xbutton.button == Button3)
        {
           printf("Button3\n");
        }
        else if (cb->event->xbutton.button == Button2)
        {
           printf("Button2\n");
        }
        else if (cb->event->xbutton.button == Button1)
        {
            Vessel::Px0 = xpos;
            Vessel::Py0 = ypos;
            printf("%d,%d\n", xpos, ypos);
            GLwDrawingAreaMakeCurrent(baseWidget(), _glxContext);
            int which = Vessel::search(w);
            printf(" WHICH=%d\n", which);

            if(which >= 0)
              ((MagicBB *)_parent) -> vessel_info(which, meas_diameter[which]);

        }
    }
    else if (cb->event->type == ButtonRelease)
    {
        if (cb->event->xbutton.button == Button3)
        {
                    printf(" R Button3\n");
        }
        else if (cb->event->xbutton.button == Button2)
        {
           printf(" R Button2\n");

        }
        else if (cb->event->xbutton.button == Button1)
        {

          printf(" R Button1\n");
        }
    }


    //---- End editable code block: ModelDraw input

}    // End ModelDraw::input()


void ModelDraw::resize ( Widget w, XtPointer callData )
{
    //---- Start editable code block: ModelDraw resize

    XmDrawingAreaCallbackStruct *cbs = (XmDrawingAreaCallbackStruct*) callData;

    //--- Comment out the following line when ModelDraw::resize is implemented:

    //::VkUnimplemented ( w, "ModelDraw::resize" );

    printf(" resize\n");


    //---- End editable code block: ModelDraw resize

}    // End ModelDraw::resize()
```

```
void ModelDraw::display(int x, int y)
{
    GLwDrawingAreaMakeCurrent(baseWidget(), _glxContext);
    glMatrixMode(GL_COLOR);
    glRasterPos2i(x,  y);

   // glDrawPixels(_width, _height, GL_LUMINANCE, GL_UNSIGNED_BYTE, (GLvoid *)(*_grayIn
    glFlush();
}

void ModelDraw::ginit ( Widget wid, XtPointer callData )
{
    //---- Start editable code block: ModelDraw expose

    XmDrawingAreaCallbackStruct *cbs = (XmDrawingAreaCallbackStruct*) callData;

    //--- Comment out the following line when ModelDraw::expose is implemented:

    //::VkUnimplemented ( w, "ModelDraw::expose" );

    init();
    init_display();
    //clear_display();
    //display();

    //---- End editable code block: ModelDraw expose

}    // End ModelDraw::expose()


void ModelDraw::expose ( Widget wid, XtPointer callData )
{
    //---- Start editable code block: ModelDraw expose

    XmDrawingAreaCallbackStruct *cbs = (XmDrawingAreaCallbackStruct*) callData;

    //--- Comment out the following line when ModelDraw::expose is implemented:

    //::VkUnimplemented ( w, "ModelDraw::expose" );

    //printf(" ModelDraw::expose \n");
    if(_init_OpenGL == 0)
      init_display();
    //if(_init_OpenGL == 1)
      clear_display();
    display();

    //---- End editable code block: ModelDraw expose

}    // End ModelDraw::expose()


void ModelDraw::input ( Widget w, XtPointer callData )
{
    //---- Start editable code block: ModelDraw input

    XmDrawingAreaCallbackStruct *cb = (XmDrawingAreaCallbackStruct*) callData;

    //--- Comment out the following line when ModelDraw::input is implemented:

    //::VkUnimplemented ( w, "ModelDraw::input" );

    int xpos = cb->event->xmotion.x;
    int ypos = cb->event->xmotion.y;
```

```
}

void ModelDraw::set(int w, int h, unsigned char **grayimg)
{
  _width = w;
  _height = h;
  _grayImg = grayimg;
}

const char * ModelDraw::className() // classname
{
    return ("ModelDraw");
} // End className()

void ModelDraw::init_OpenGL()
{
    printf(" init_OpenGL\n");
    Display *dpy = XtDisplay(baseWidget());
    XVisualInfo *visionInfo;
    int attribs[] = { GLX_RGBA, 0};

    if (!(visionInfo = glXChooseVisual(dpy, DefaultScreen(dpy), attribs)))
    {
      //cerr << "Error: no suitable RGB visual" << endl;
      //exit(EXIT_FAILURE);
      return;
    }

    _glxContext = glXCreateContext(dpy, visionInfo, 0, GL_TRUE);
    GLwDrawingAreaMakeCurrent(baseWidget(), _glxContext);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glViewport(0, 0, get_widthUI(), get_heightUI());
    glOrtho(0, get_widthUI(),  0, get_heightUI(), 0, 10.0);

    _init_OpenGL = 1;
    printf(" init_OpenGL dn\n");

    XFontStruct *font = XLoadQueryFont(dpy,
                          "-adobe-courier-medium-r-normal--14-*-*-*-*-*-*-*");
    glXUseXFont(font->fid, 32, 96, 2000+32);
}

void ModelDraw::clear_display()
{
  GLwDrawingAreaMakeCurrent(baseWidget(), _glxContext);
  glClear(GL_COLOR_BUFFER_BIT);
}

void ModelDraw::init_display()
{
    init_OpenGL();
}

void ModelDraw::display()
{
    GLwDrawingAreaMakeCurrent(baseWidget(), _glxContext);

    //glDrawPixels(_width, _height, GL_LUMINANCE, GL_UNSIGNED_BYTE, (GLvoid *)(*_grayIn

    Vessel::readFile();
    Vessel::drawAll();
    glFlush();
}
```

```
#include "ModelDraw.h"
#include <Vk/VkEZ.h>
#include <Xm/DrawingA.h>
#include <Vk/VkResource.h>
#include <Vk/VkSimpleWindow.h>
#include "Vessel.h"
#include "LinkList.h"

//---- Start editable code block: headers and declarations

#include <GL/GLwMDrawA.h>
#include <stdio.h>
#include "ImgBase.h"
#include "MagicBB.h"

char   _saveFile[300];

//---- End editable code block: headers and declarations



//---- ModelDraw Constructor


ModelDraw::ModelDraw(int x, int y, int w, int h, const char *name,
   Widget parent) :  DrawingAreaUI(x, y, w, h, name, parent)
{
    // This constructor calls ModelDrawUI(parent, name)
    // which calls ModelDrawUI::create() to create
    // the widgets for this component. Any code added here
    // is called after the component's interface has been built

    //---- Start editable code block: ModelDraw constructor

    //---- End editable code block: ModelDraw constructor



}    // End Constructor


ModelDraw::~ModelDraw()
{
    // The base class destructors are responsible for
    // destroying all widgets and objects used in this component.
    // Only additional items created directly in this class
    // need to be freed here.

    //---- Start editable code block: ModelDraw destructor

    clear_memory();

    //---- End editable code block: ModelDraw destructor

}    // End Destructor

void ModelDraw::clear_memory()
{
   delete _grayImg;
   delete *_grayImg;
   _grayImg = NULL;
}

void ModelDraw::set(unsigned char **grayimg)
{
  _grayImg = grayimg;
```

```
    static Vessel *vessel[158];

    // PreVessel number
    static int preI;

public:
    // Constructor
    Vessel(int flag, int x1, int y1, double d1, double d2, double a1,
        double a2, LinkList * list);

    // destructor
    ~Vessel()
    {
        delete linkList;
        delete leftList;
        delete rightList;
        delete centrList;
    }

    // Move parallelly the coordinates of the vessel
    void move(int x, int y, Widget canvas, GC gc);

    // Draw the vessel on the canvas in the window
    void draw();
    void draw(float, float, float);


    // Draw all the vessels
    static void drawAll();

    // Read vessels from data file
    static int readFile();

    static int search(Widget canvas);

    static int ifInRange(int, int, int, int, int, int, int, int, int, int);

    static int ifRight(int, int, int, int, int, int);

    static void pressPoint(Widget w, XtPointer, XEvent *event, char*);

    static void drawCondition(int, Widget, GC);

    static void addEdge(LinkList *, int, int, int, int);

    static void fillingVessel(Widget,  int , int);

    static void filling(int, int, GC);

    void finding(int, int, LinkList *);

    void changeText(int, GC);

    static int point[724][965];

    static int Px0, Py0;

};

#endif  // VESSEL_H
```

```
#ifndef VESSEL_H
#define VESSEL_H

/*
 * File:        Vessel.h
 *
 * Author:      Ying Dai
 *              ydai@eecs.uic.edu
 *
 * Description:
 *      The file contains the definition of the Vessel class.
 */

#include <X11/Intrinsic.h>
#include "LinkList.h"

/*
 * Class:       Vessel
 *
 * Description:
 *      This class contains the coordinates of the start point, the length of
 *      the start diameter and the length of the end diameter, the angle of the
 *      the start line to X axis and the angle of the end line to X axis, and
 *      the linkList which contains the coordinates of the points on the center
 *      axis of the vessel.
 */

class Vessel
{
private:
    int flag;     //0 means not draw up and down line
                  //1 means to draw up line
                  //2 means to draw down line
                  //3 means to draw both line
                  //4 special line
    int upX;     // x coordinate of the start point
    int upY;     // y coordinate of the start point
    double upLength;     // the length of the start diameter
    double downLength;   // the length of the end diameter
    double upAngle;      // the angle of the up line to the X axis
    double downAngle;    // the angle of the down line to the X axis
    LinkList *linkList;          // the points in the center axis of vessel
    LinkList *leftList;          // the points in the left axis of vessel
    LinkList *rightList;         // the points in the right axis of vessel
    LinkList *edgePointList;     // all points in the edges of the vessel;
    LinkList *includePointList; // all points inside of the vessel;
    LinkList *centrList;         // the points in the center axis of vessel

    // Get some points(x,y) in the left line/edge  of the vessel to build
    // up the left linkList and get some points (x,y) in the right line/
    // edge of the vessel to build up the right linkList
    void linkLists(LinkList &leftLinkList, LinkList &rightLinkList);

    //
    void addAllEdge(LinkList *);

    //
    void rangeOfVessel(LinkList *);

    // Number of vessels in the data file
    static int vesselNum;

    // Max number of vessels supported
    static int maxVesselNum;

    // Vessel array
```

```
    while (p2 -> getNext() != 0)
    {
//      XDrawPoint(Draw::display, Draw::window, tempGC, p2->getX(),
//      p2->getY());
        glBegin(GL_POINTS);
            glVertex2f(p2->getX(), p2->getY());
        glEnd();
        p2 = p2->getNext();
    }
    //XDrawPoint(Draw::display, Draw::window, tempGC, p2->getX(),
    //p2->getY());
    glBegin(GL_POINTS);
        glVertex2f(p2->getX(), p2->getY());
    glEnd();
    delete p1;
    delete p2;

}
```

```
    }
}
```

```
    void
Vessel::filling(int x, int y, GC gc)
{
    if ( point[x][y] == 0)
    {
        point[x][y] = 2;
        glBegin(GL_POINTS);
            glVertex2f(x, y);
        glEnd();

        //XDrawPoint(Draw::display, Draw::window, gc, x, y);
        filling(x, y-1, gc);
        filling(x, y+1, gc);
        filling(x-1, y, gc);
        filling(x+1, y, gc);
    }
}
```

```
   void
Vessel::fillingVessel(Widget canvas, int i, int j)
{
    int k, l, m, n, temp;
    GC tempGC;
/*    if (j == 0 )
        tempGC = Draw::whiteGC;
    if (j == 1)
        tempGC = Draw::redGC;
    if (j == 2)
        tempGC = Draw::blueGC;
    if (j == 3)
        tempGC = Draw::yellowGC;
    if (j == 4)
        tempGC = Draw::greenGC;
    if (j == 5)
        tempGC = Draw::blackGC;*/
    if (Vessel::vessel[i-1]->includePointList->getHead() == 0)
        Vessel::vessel[i-1]->rangeOfVessel(Vessel::vessel[i-1]->
                            includePointList);
    Node *p1, *p2;

    p1 = Vessel::vessel[i-1]->edgePointList->getHead();
    p2 = Vessel::vessel[i-1]->includePointList->getHead();
    while (p1 -> getNext() != 0)
    {
        glBegin(GL_POINTS);
            glVertex2f(p1->getX(), p1->getY());
        glEnd();
        //XDrawPoint(Draw::display, Draw::window, tempGC, p1->getX(),
//      p1->getY());
        p1 = p1->getNext();
    }
 //   XDrawPoint(Draw::display, Draw::window, tempGC, p1->getX(),
  //  p1->getY());
    glBegin(GL_POINTS);
        glVertex2f(p1->getX(), p1->getY());
    glEnd();
```

```
        }
    }

    else
    {
        if (x1 > x2)
        {
            temp = x1;
            x1 = x2;
            x2 = temp;
            temp = y1;
            y1 = y2;
            y2 = temp;
        }

        int preX = x1;
        int preY = y1;
        for ( x = x1; x <= x2; x++)
        {
            y = (y2 - y1)*(x - x1)/(x2 - x1) + y1;
            if ((y2 - y1) * (x - x1) % (x2 - x1) != 0)
            {
                list->add(preX, y);
            }
            if ( y != (preY + 1) && y != (preY -1) && y != preY)
            {
                if (preY > y )
                {
                    for ( temp = preY-1; temp > y; temp--)
                    {
                        list->add(preX, temp);
                    }
                }
                else
                {
                    for ( temp = preY+1; temp < y; temp++)
                    {
                        list->add(preX, temp);
                    }
                }

            }


            list->add(x, y);
            preX = x;
            preY = y;
        }
    }
}



    void
Vessel::finding(int x, int y, LinkList *list)
{
    if ( point[x][y] == 0)
    {
        point[x][y] = 2;
        list ->add(x, y);
        finding(x, y-1, list);
        finding(x, y+1, list);
        finding(x-1, y, list);
        finding(x+1, y, list);
```

```
            {
                if(point[[redacted]0] == 1)
                    flag2[redacted];
            }

            int flag3 = 0;
            int flag4 = 0;
            for (k = 0; k < y0; k++)
            {
                if(point[x0][k] == 1)
                    flag3 ++;
            }
            for (k = y0 + 1; k <= MODEL_WIDTH; k++)
            {
                if (point[x0][k] == 1 )
                    flag4 ++;
            }
            if (flag1 != 0  && flag2 != 0 && flag3 != 0 && flag4 != 0 &&
                point[x0][y0] == 0)
            {
                Vessel::finding(x0, y0, list);
            }
        }
    }


    delete p1;
    delete p2;
    delete p3;
    }
}

    void
Vessel::addEdge(LinkList * list, int x1, int y1, int x2, int y2)
{
    int x, y;
    int temp;

    if ( x1 == x2 && y1 == y2)
        list->add(x1, y1);

    else if ( x2 == x1)
    {
        if ( y1 > y2 )
        {
            temp = y1;
            y1 = y2;
            y2 = temp;
        }
        for ( temp = y1; temp <= y2; temp++)
        {
            list->add(x1, temp);
        }
    }

    else if ( y2 == y1)
    {
        if ( x1 > x2 )
        {
            temp = x1;
            x1 = x2;
            x2 = temp;
        }
        for ( temp = x1; temp <= x2; temp++)
        {
            list->add(temp, y1);
```

```
    y2 = rightList->getHead()->getY();
    x3 = leftList->getTail()->getX();
    y3 = leftList->getTail()->getY();
    x4 = rightList->getTail()->getX();
    y4 = rightList->getTail()->getY();
    addEdge(list, x1, y1, x2, y2);
    addEdge(list, x3, y3, x4, y4);
    leftList->setCurve(list);
    rightList->setCurve(list);
}

    void
Vessel::rangeOfVessel(LinkList *list)
{
    int i, j;
    for ( i = 0; i < 724; i++)
    {
        for (j = 0; j < 965; j++)
        {
            point[i][j] = 0;
        }
    }
    if(linkList->getHead()->getX() != 0)
    {
    int x0, y0, flag = 0;
    int minX, minY, maxX, maxY;
    int tempX, tempY;
    int i, j, k;
    Node *p1, *p2;
    Node *p3;
    p3 = edgePointList->getHead();
    while (p3->getNext() != 0)
    {
        point[p3->getX()][p3->getY()]= 1;
        p3 = p3->getNext();
    }
    point[p3->getX()][p3->getY()]= 1;
    p1 = edgePointList->getHead();
    p2 = edgePointList->getTail();
    minX = maxX = p2->getX();
    minY = maxY = p2->getY();
    while (p1 ->getNext() != 0)
    {
        if (p1->getX() > maxX)
            maxX = p1->getX();
        if (p1->getX() < minX)
            minX = p1->getX();
        if (p1->getY() >maxY)
            maxY = p1->getY();
        if (p1->getY() < minY)
            minY = p1->getY();
        p1 = p1->getNext();
    }

    for ( x0 = minX; x0 <= maxX && flag == 0; x0++)
    {
        for (y0 = minY; y0<= maxY && flag == 0; y0++)
        {
            int flag1 = 0;
            int flag2 = 0;
            for (k = 0; k < x0; k ++)
            {
                if(point[k][y0] == 1 )
                    flag1 ++;
            }
            for ( k = x0+1; k <= MODEL_HEIGHT; k++)
```

```
{
   // the point is on the right of all four lines
   if(ifRight(x1, y1, x2, y2, x, y)&&
      ifRight(x2, y2, x4, y4, x, y)&&
      ifRight(x4, y4, x3, y3, x, y)&&
      ifRight(x3, y3, x1, y1, x, y))
        return 1;

   if(!ifRight(x1, y1, x2, y2, x, y)&&
      !ifRight(x2, y2, x4, y4, x, y)&&
      !ifRight(x4, y4, x3, y3, x, y)&&
      !ifRight(x3, y3, x1, y1, x, y))
        return 1;

    // the point is out of range
    else
        return 0;
}


/*
 * Name:        Vessel::ifRight
 *
 * Description:
 *      Whether or not the point (x3, y3) is on the right of the line.
 *      The line is from (x1, y1) to (x2, y2).
 *
 *
 * Parameter:
 *      int x1
 *      int y1
 *      int x2
 *      int y2
 *      int x3
 *      int y3
 *
 * Return value:
 *      1       Success
 *      0       Failure
 */

    int
Vessel::ifRight(int x1, int y1, int x2, int y2, int x3, int y3)
{
    int a1 = x2 - x1;
    int a2 = y2 - y1;
    int b1 = x3 - x1;
    int b2 = y3 - y1;
    if ( (a1 * b2 - a2 * b1) > 0 )
        return 1;
    else
        return 0;
}



    void
Vessel::addAllEdge(LinkList *list)
{

    int x1, y1, x2, y2;
    int x3, y3, x4, y4;
    x1 = leftList->getHead()->getX();
    y1 = leftList->getHead()->getY();
    x2 = rightList->getHead()->getX();
```

```
                    }
                    glVer[illegible]2f(p4->getX(), MODEL_HEIGHT-p[illegible]getY());
                    glEnd();
                    glFlush();
                    */
                    //XDrawPoint(Draw::display, Draw::window,
                    //Draw::greenGC, p4->getX(), p4->getY() );

                    //Vessel::filling(Px0, Py0, Draw::redGC);
                    //Vessel::vessel[i]->changeText(i, Draw::whiteGC);

                    //return i + 1;
                }
                p1 = p1->getNext();
                p2 = p2->getNext();
            }
        }
        i ++;
    }

    // the point is out of all vessels
    if (flagIn == 0)
    {
        // keep all the vessels in black.
        if (preI != -1)
        {
            //Vessel::vessel[preI]->changeText(preI, Draw::blackGC);
            Vessel::vessel[preI]->draw();
            preI = -1;
        }
    }
    return 0;
}

/*
 * Name:        Vessel::ifInRange
 *
 * Description:
 *      Whether or not the point (x, y) is in the range. The range with
 *      four edge lines.  The first is from (x1, y1) to (x2, y2), the
 *      second is from (x2, y2) to (x3, y3), the third is from (x3, y3)
 *      to (x4, y4), and the fourth is from (x4, y4) to (x1, y1).
 *
 *
 *
 * Parameter:
 *      int x1
 *      int y1          // point(x1, y1)
 *      int x2
 *      int y2          // point(x2, y2)
 *      int x3
 *      int y3          // point(x3, y3)
 *      int x4
 *      int y4          // point(x4, y4)
 *      int x
 *      int y
 *
 *
 * Return value:
 *      1       Success
 *      0       Failure
 */

    int
Vessel::ifInRange(int x1, int y1, int x2, int y2, int x3, int y3, int x4,
                int y4, int x, int y)
```

```
/*
 * File:                V[redacted]el.C
 *
 * Author:              Ying Dai
 *                      ydai@eecs.uic.edu
 *
 * Description:
 *      This file contains the implementation of the Vessel class.
 */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "Vessel.h"
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/cursorfont.h>
#include <Xm/Xm.h>
#include <GL/gl.h>
#include <GL/glu.h>

#include "ModelDraw.h"

// Initialize static variables.
int Vessel::Px0, Vessel::Py0;
int Vessel::vesselNum = 0;
int Vessel::maxVesselNum = 158;
int Vessel::point[724][965];
Vessel *Vessel::vessel[158];
int Vessel::preI = -1;

/*
 * Name:        Vessel::Vessel
 *
 * Description:
 *      Constructor
 *
 */

Vessel::Vessel(int flag1, int x1, int y1, double d1, double d2, double a1,
    double a2, LinkList * list)
    :   flag(flag1),
        upX(x1),
        upY(y1),
        upLength(d1),
        downLength(d2),
        upAngle(a1),
        downAngle(a2),
        linkList(list)
{
    leftList = new LinkList();
    rightList = new LinkList();
    edgePointList = new LinkList();
    includePointList = new LinkList();
    linkLists(*leftList, *rightList);
    centrList = linkList->absLinkList(upX, upY); // debug
    addAllEdge(edgePointList);
//    rangeOfVessel(includePointList);
}


/*
 * Name:        Vessel::move
 *
 * Description:
 *      Move parallelly the coordinates of the vessel.
```

```
 *
 * Parameters:
 *      int x   X coordinate of the new location
 *      int y   Y coordinate of the new location
 *
 * Return value:
 *      None
 */

    void
Vessel::move(int x, int y, Widget canvas, GC gc)
{
    upX = x;
    upY = y;

    // Reate the left and the right lists of the vessel.
    delete leftList;
    delete rightList;
    leftList = new LinkList();
    rightList = new LinkList();
    linkLists(*leftList, *rightList);

    // Redraw the vessel.
    draw();
}


/*
 * Name:        Vessel::linkLists
 *
 * Description:
 *      Appends left points and right points in the left linkList and right
 *      linkList.
 *
 * Parameters:
 *      LinkList &leftLinkList  left empty linkList
 *      LinkList &rightLinkList right empty linkList
 *
 * Return value:
 *      None
 */

void Vessel::linkLists(LinkList &leftLinkList, LinkList &rightLinkList)
{
    int x0, y0;         // the original point

    int j;
    double l = upLength/2;      // l is the half length of the up line
    double a = upAngle;         // start angle

    int i = linkList->getLength();

    double deltaL, deltaX, deltaY, deltaA;
    deltaL = (downLength/2 - upLength/2)/(i-1);
    deltaA = (downAngle - upAngle)/(i-1);
    LinkList *list;
    list = linkList->absLinkList(upX, upY);
    Node *p;
    p = list->getHead();        // p points to the head
                                // of the linkList

    // All the points in the linkList are original points to calculate the
    // left points and right points of them. All left points builds up the
    // left linkList and right points builds up the right linkList
    if (flag != 4)
    {
```

```
        while ( p != 0 )
        {
            x0 = p->getX();
            y0 = p->getY();
            deltaX = l*cos(a);
            deltaY = l*sin(a);
            leftLinkList.add(int(x0 - deltaX), int(y0 - deltaY));
            rightLinkList.add(int(x0 + deltaX), int(y0 + deltaY));
            p = p->getNext();
            l += deltaL;
            a += deltaA;
        }
        delete list;
    }

    if (flag == 4)
    {
        j = 0;
        while ( p != 0 )
        {
            x0 = p->getX();
            y0 = p->getY();
            i = 15;
            deltaX = l*cos(a);
            deltaY = l*sin(a);
            leftLinkList.add(int(x0 - deltaX ), int(y0 - deltaY));
            rightLinkList.add(int(x0 + deltaX), int(y0 + deltaY ));
            p = p->getNext();
            j ++;
            if (j > 2)
                l = l -i;
            else
                l += i;
            a += deltaA;
        }
        delete list;
    }
    // Calculate the forward difference array.
    leftLinkList.process();
    rightLinkList.process();
}


/*
 * Name:        Vessel::draw
 *
 * Description:
 *      Draw the vessel in different color on the canvas in the window.
 *      Draw::gc        Black
 *      Draw::redGC     Red
 *      Draw::whiteGC   White
 *
 * Parameters:
 *      Widget canvas
 *      GC     gc
 *
 * Return value:
 *      None
 */

    void
Vessel::draw()
{
    // Draw start line.
        if (flag == 1 || flag == 3 || flag == 4)
            glBegin(GL_LINES);
```

```
            glVertex2f(leftList->getHead()->getX(), MODEL_HEIGHT-leftList->
            getHead()->getY());
            glVertex2f(rightList->getHead()->getX(), MODEL_HEIGHT-rightList->
            getHead()->getY());
        glEnd();
        //XDrawLine(Draw::display, Draw::window, gc,
        //leftList->getHead()->getX(), leftList->getHead()->getY(),
        //rightList->getHead()->getX(), rightList->getHead()->getY());

    // Draw end line.
        if (flag == 2 || flag == 3 || flag == 4)
        glBegin(GL_LINES);
            glVertex2f(leftList->getTail()->getX(), MODEL_HEIGHT-leftList->
            getTail()->getY());
            glVertex2f(rightList->getTail()->getX(), MODEL_HEIGHT-rightList->
            getTail()->getY());
        glEnd();
        //XDrawLine(Draw::display, Draw::window, gc,
        //leftList->getTail()->getX(), leftList->getTail()->getY(),
        //rightList->getTail()->getX(), rightList->getTail()->getY());

    // Draw left and right curves.
    // centrList->draw(canvas, gc); // debug
    // leftList->draw(canvas, gc);
    // rightList->draw(canvas, gc);
    leftList->drawCurve();
    rightList->drawCurve();
}

void
Vessel::draw(float r, float g, float b)
{
    // Draw start line.
        if (flag == 1 || flag == 3 || flag == 4)
        glColor3f(r, g, b);
        glBegin(GL_LINES);
            glVertex2f(leftList->getHead()->getX(), MODEL_HEIGHT-leftList->
            getHead()->getY());
            glVertex2f(rightList->getHead()->getX(), MODEL_HEIGHT-rightList->
            getHead()->getY());
        glEnd();
        //XDrawLine(Draw::display, Draw::window, gc,
        //leftList->getHead()->getX(), leftList->getHead()->getY(),
        //rightList->getHead()->getX(), rightList->getHead()->getY());

    // Draw end line.
        if (flag == 2 || flag == 3 || flag == 4)
        glBegin(GL_LINES);
            glVertex2f(leftList->getTail()->getX(), MODEL_HEIGHT-leftList->
            getTail()->getY());
            glVertex2f(rightList->getTail()->getX(), MODEL_HEIGHT-rightList->
            getTail()->getY());
        glEnd();
        //XDrawLine(Draw::display, Draw::window, gc,
        //leftList->getTail()->getX(), leftList->getTail()->getY(),
        //rightList->getTail()->getX(), rightList->getTail()->getY());

    // Draw left and right curves.
    // centrList->draw(canvas, gc); // debug
    // leftList->draw(canvas, gc);
    // rightList->draw(canvas, gc);
    leftList->drawCurve(r, g, b);
    rightList->drawCurve(r, g, b);
}

/*
```

```
 * Name:        Vessel::drawAll
 *
 * Description:
 *      This static function draws all the vessels on the canvas
 *
 * Parameters:
 *      Widget canvas
 *
 * Return value:
 *      None
 */

    void
Vessel::drawAll()
{
    int i;

    for (i = 0; i < Vessel::vesselNum; i++)
    {
        if ( i != 85 && i != 87 && i != 91 && i != 92 && i != 93 && i != 94
        && i != 95 && i != 96 && i != 97 && i != 98 && i != 101 && i != 102
        && i != 106 && i != 107 && (Vessel::vessel[i] -> upLength != 0 ||
        Vessel:: vessel[i] -> downLength != 0))
        {
            if(meas_diameter[i] == 0.0)
              Vessel::vessel[i]->draw(1, 1, 1);
            else
              Vessel::vessel[i]->draw(0, 0, 1);
        }
    }
}

    void
Vessel::changeText(int i, GC gc)
{
   /* XTextItem line[1];
    char s[4];
    sprintf(s, "%d", i+1);
    line[0].chars = s;
    line[0].nchars = strlen(s);
    line[0].font = XLoadQueryFont(Draw::display,
    "*courier-bold-r*140*") -> fid;
        XDrawString(Draw::display, Draw::window, gc,
        Vessel::vessel[i]->centrList->getHead()->getNext()->
        getNext()->getX(),
        Vessel::vessel[i]->centrList->getHead()->getNext()->
        getNext()->getY(),
        line[0].chars, line[0].nchars);
*/
}

/*
 * Name:        Vessel::readFile
 *
 * Description:
 *      Read the data file and create the vessel list.
 *
 * Parameter:
 *      const char *fileName    Name of the data file
 *
 * Return value:
 *      0       Success
 *      -1      Failure
 */

    int
```

```
Vessel::readFile()
{
    FILE *fp;
    FILE *fq;
    int flag1;
    int i, j, num;
    int x, y;
    int d3, d4;
    float a1, a2, d1, d2;

    if (!(fp = fopen("original.data", "r")))
    {
        printf("Cannot open %s.\n", "original.data");
        return -1;
    }

    if (!(fq = fopen("Test.data", "r")))
    {
        printf("Cannot open %s.\n", "Test.data");
        return -1;
    }

    if (vesselNum)
    {
        for (i = 0; i < vesselNum; i++)
            delete vessel[i];

        // Clear the drawing area.
    }

    vesselNum = 0;
    do
    {
        LinkList * list = new LinkList();
        fscanf(fp, "%d", &flag1);

        for (i = 0; i < 5; i++)
        {
            fscanf(fp, "%d%d%",&x, &y);
            list->add(x,y);
        }

        fscanf(fp, " %f %f\n",  &a1, &a2);

        fscanf(fq, "%d %f %f\n", &num, &d1, &d2);
        if (d1 > 3)
            d1 = 3;
        if (d1 < 0.03 && d1 != 0)
            d1 = 0.03;
        if (d2 > 3)
            d2 = 3;
        if (d2 < 0.03 && d2 != 0)
            d2 = 0.03;
        d1 = 40 * d1;
        d2 = 40 * d2;
        d3 = d1;
        d4 = d2;

        vessel[vesselNum] = new Vessel(flag1, list->getHead()->getX(),
            list->getHead()->getY(), d3, d4, a1, a2, list);
        vesselNum++;
    } while (!feof(fp) && vesselNum < maxVesselNum);

    for ( i = 0; i < 724; i++)
    {
        for (j = 0; j < 965; j++)
```

```
            point[i][j] = 0;
    }

    fclose(fq);
    fclose(fp);
    return 0;
}

/*
 * Name:        Vessel::pressPoint
 *
 * Description:
 *      Use the mouse to press one point in the vessel drawArea, find this
 *      point is in which vessel, then change the vessel in red.
 *
 * Parameter:
 *      Widget w
 *      Xtpointer
 *      XEvent*
 *      char*
 *
 * Return value:
 *     None
 */

void Vessel::pressPoint(Widget w, XtPointer, XEvent *event, char*)
{
    int i, j;
    for ( i = 0; i < 724; i++)
    {
        for (j = 0; j < 965; j++)
        {
            if (point[i][j] == 2)
                glBegin(GL_POINTS);
                    glVertex2f(i, j);
                glEnd();

                //XDrawPoint(Draw::display, Draw::window, Draw::blackGC, i,j);

            if (point[i][j] != 0)
                point[i][j] = 0;
        }
    }


    if (Vessel::vessel[0] != 0)
        i = Vessel::search(w);
}


/*
 * Name:        Vessel::search
 *
 * Description:
 *      Find the click point is in which vessel. and change that vessel
 *      in red.
 *
 * Parameter:
 *      Widget canvas
 *
 *
 * Return value:
 *      1       success
 *      0       failse
 *
 */
```

```
int
Vessel::search(Widget can
{
    int flagIn = 0;
    int i = 0;
    int j;
    printf(" P** %d %d\n",Px0,Py0);
    while (i < Vessel::vesselNum && flagIn == 0)
    {
        if (i != 85 && i != 87 && i != 91 && i != 92 && i != 93 && i != 94
            && i != 95 && i != 96 && i != 97 && i != 98 && i != 101 && i != 102
            && i != 106 && i != 107 && (Vessel::vessel[i] ->upLength != 0 ||
            Vessel::vessel[i] ->downLength != 0))
        {
            Node *p1, *p2;
            p1 = Vessel::vessel[i]->leftList->setList()->getHead();
            p2 = Vessel::vessel[i]->rightList->setList()->getHead();
            while ( p1->getNext() != 0 && p2->getNext() != 0&& flagIn == 0)
            {
                // find the vessel include the point
                if(ifInRange(p1->getX(), p1->getY(), p1->getNext()->getX(),
                    p1->getNext()->getY(), p2->getX(), p2->getY(),
                    p2->getNext()->getX(), p2->getNext()->getY(),
                    Px0, Py0))
                {
                    flagIn = 1;

                    // change the pre found vessel in black
                    if (preI != -1)
                    {
                        //Vessel::vessel[preI]->changeText(preI, Draw::blackGC);
                        //Vessel::vessel[preI]->draw();
                     }

                    // change the found vessel in red
                    Vessel::vessel[i]->draw(1.0, 0.0, 0.0);
                    //Vessel::vessel[i]->draw();
                    preI = i;
                    printf(" vessel:  %d \n", preI);
                    return preI;

                    /*
                    Node *p3;
                    p3 = Vessel::vessel[i]->edgePointList->getHead();
                    while (p3->getNext() != 0)
                    {
                        point[p3->getX()][p3->getY()]= 1;
                        p3 = p3->getNext();
                    }
                    point[p3->getX()][p3->getY()]= 1;

                    if (Vessel::vessel[i]->includePointList->getHead() == 0)
                        Vessel::vessel[i]->rangeOfVessel(Vessel::vessel[i]->
                                        includePointList);
                    Node *p4;
                    p4 = Vessel::vessel[i]->includePointList->getHead();
                    glBegin(GL_POINTS);
                    glColor3f(1.0, 0.0, 0.0);
                    while (p4->getNext() != 0)
                    {
                        //XDrawPoint(Draw::display, Draw::window,
                //      Draw::greenGC, p4->getX(), p4->getY() );
                        glVertex2f(p4->getX(), MODEL_HEIGHT-p4->getY());
                        //printf(" %d %d *** \n",p4->getX(),p4->getY());
                        p4 = p4->getNext();
```